UNIVERSITY OF TURKU

ABSTRACT

	Bachelor's thesis
x	Master's thesis
	Licentiate's thesis
	Doctor's thesis

Subject	Information Systems Science	Date	1.6.2019
Author(s)	Vilma Blomberg	Student number	509426
		Number of pages	84
Title	Adopting DevOps Principles, Practices and Tools. Case: Identity & Access Management		
Supervisor(s)	Prof. Dr. Hannu Salmela, Prof. Dr. Anne-Francoise Rutkowski		

DevOps is a framework that emerged to build a bridge between IT Development and Operations. DevOps combines Agile and Lean methodologies with a purpose of creating seamless workflow from development to operations using Continuous Integration, Continuous Delivery and Continuous Feedback mechanisms. Adopting DevOps has been of interest for many organizations and practitioners for a while now due to its various benefits for business. However, there is a lack of knowledge and understanding on what is meant by DevOps when it comes to the key concepts, practices, tools, and the benefits and challenges of DevOps adoption. Organizations and teams are missing guidance on how to adopt DevOps in their specific context. This design science research is conducted to understand how to adopt DevOps principles, practices and tools in the Identity and Access Management of a large multinational corporation. The result of this study are the proposed models for adopting DevOps, including the formation of the teams and the processes covering build, test and deployment of identity management system (SailPoint IIQ) and onboarding new applications to the system. Three design artifacts are built and evaluated against identified problem areas in DevOps adoption, providing insights to the research community and industry practitioners.

Key words	DevOps, IAM, Identity Management, Continuous Integration, Continuous Deployment, Continuous Feedback, Agile, SailPoint, automation, software development
Further information	

ADOPTING DEVOPS PRINCIPLES, PRACTICES AND TOOLS

Case: Identity & Access Management

Master´s Thesis
in Information Systems Science

Author:
Vilma Blomberg

Supervisors:
Prof. Dr. Hannu Salmela
Prof. Dr. Anne-Francoise Rutkowski

1.6.2019
Turku

The originality of this thesis has been checked in accordance with the University of Turku quality assurance system using the Turnitin OriginalityCheck service.

Table of contents

List of figures

List of tables

ACKNOWLEDGMENTS

I am writing these acknowledgments before submitting this paper to the Tilburg University (Netherlands), University of Turku (Finland) and Aix-Marseille University (France) for review. I want to acknowledge people involved in this thesis both from the university and the case company. Thank you for my 1st supervisor Prof. Dr. Hannu Salmela from University of Turku for the time he has dedicated and the valuable feedback he gave to me. I am also thankful for the case company providing all the means to experiment and learn.

There were number of people I met during the case study in the company, many of who have contributed to this research by taking part of the IDM implementation project, or providing their views and insights during our discussions. Above all, I am highly thankful for my mentor and advisor in the case company. It is not often that you encounter such visionary who is so deeply interested and passionate about his work. Our discussions always teach me something new. It has been a pleasure to work together.

For me, this moment is not only about writing the final words of my Master's thesis but also a moment that wraps up a two-year long chapter of my life with intense studying, working and traveling from one country to another with people from cohort 11 in the International Master in Management of IT (IMMIT) program. Finally, it is time to settle for a while and thank for my family and friends who were there for me during this hectic time at studies, at work and on my free-time.

Vilma Blomberg
1st of June 2019, Finland

1 INTRODUCTION

1.1 Background

We are living in a world were technology and speed dictates our lives on a daily basis. Companies have come to realize that regardless of the industry, technology could be used to differentiate and acquire new customers or markets. (Haffke et al. 2017, 102; Horlach et al. 2017, 5420.) Many traditional industries are faced with technological disruption that challenges the existing business models. IT and business professionals today see that continuous innovation and leanness are required to stay competitive.

To deliver value and innovate at faster speed, IT teams and organizations have started to shift from traditional waterfall model -based development methodologies towards agile, iterative development. As a result, development teams have begun to deliver IT services at such a fast pace that the operations teams are not able to keep up. DevOps is a framework that emerged to build a bridge between the "Dev", also referred to as the explorative agile IT, and "Ops", also referred to as the predictive 'keep the business running' IT. DevOps combines the Agile and Lean methodologies with a purpose of creating seamless workflow from development to operations, by minimizing any friction using different DevOps practices and tools. The silos are broken down by combining the expertise and building smaller, efficient cross-functional teams that follow the three main principles of DevOps: optimize the workflow, create efficient feedback loops, and enforce culture of experimenting and learning. (Wiedemann 2017; Kim 2016; Balalaie et al. 2016; Ebert et al. 2016; Ravichandran et al. 2016, 6; Lwakatare et al. 2015.)

In practice, the DevOps teams can deliver value faster by establishing mechanisms for Continuous Integration, Continuous Delivery and Continuous Feedback. Faster, continuous delivery of value is reached by reducing manual work and by building automated deployment pipelines (Verona 2016, 3–5). The automated deployment pipelines are built by integrating the latest technology and tools that help speeding up the work while all the time building quality into the product. Research shows (Kim et al. 2016) that adopting DevOps results in improvements in multiple different areas, such as throughput and reliability metrics, information security, lead time and amount of deployments, mean time to recover, productivity, profitability, market share, and employee satisfaction.

However, DevOps is not a simple process that can be easily implemented as it is. Instead, it is a conceptual framework that provides principles, practices and tools that can be embedded to an organization's processes to gain benefits, but it requires the organization to restructure the way it works. (Sharma 2017, 40; Erich et al. 2014.) On an organizational level, DevOps adoption is an evolutionary journey. The journey starts with identifying the need for DevOps and starting to build the foundation for DevOps. Later, the

organization typically moves on to normalize their technology stack, standardize and scale the DevOps practices in the organization, and continue automating different pieces of the software delivery so that the IT solutions can be provided continuously in an automated, self-service manner (State of DevOps 2018).

The DevOps adoption journey is not a road without obstacles. The challenges that organizations face while adopting DevOps are both technical and cultural. It is not easy, especially in large companies, to overcome the initial "Dev" vs "Ops" mentality and restructure the organization that has for long been emphasizing governance structures and processes over culture. Change management efforts are needed to manage people that are facing radical changes in their roles and responsibilities. Also, many IT organizations are strictly following ITIL processes that fight against DevOps ideology and the aim of the continuous delivery of value. Most of the technical challenges with DevOps adoption stem from the automation and move from legacy infrastructure to infrastructure as a code. Many organizations have monolithic architectures that are tightly coupled, and DevOps adoption forces them to build more flexible architecture that allows continuous evolution.

Adopting DevOps has been of interest for many organizations and practitioners for a while now. However, there is a lack of knowledge and understanding of what is meant by DevOps when it comes to the key concepts, practices, tooling, and the main benefits and challenges of DevOps adoption (Ghantous & Gill 2017). Moreover, State of DevOps report from 2018 show that C-level executives have often overly positive views on how DevOps is adopted in their organization, while in reality they lack understanding on the DevOps adoption, its impacts, and possible challenges or bottlenecks related to it. Organizations and teams are missing guidance on how to adopt DevOps in their specific context (Gruver 2016, 16).

1.2 Introduction to the case study

Erich et al. (2014) conclude that DevOps is an artifact that must be adapted to its unique application environment. The purpose of this study is to expand the research on DevOps adoption into a new, unexplored application domain by using design science research method. The chosen application domain for this research is Identity & Access Management (IAM), which is a critical field of IT providing the building blocks for organizations' information security. Delivering value from IAM helps the organization for example to automate cumbersome processes related to identity lifecycle management, management of groups and accounts, and provisioning accesses to target applications and systems.

The research environment of this thesis is a multinational corporation, referred to as Company X, and a case study is done in the Identity and Access Management team of the

company. The goal of the IAM team is to provide digital Identity and Access Management solutions for all the applications in company (~1000 applications). The IAM team is implementing a new Identity Management -system SailPoint IIQ that will establish centralized identity management and automated access provisioning to the applications. The business value of the IAM services is measured by how many applications are IAM compliant, and the target of the IAM team is to provide their services in a self-service manner. To reach the team's goals and efficiently generate business value, the team needs to deliver its services from development to the operations at a fast speed while maintaining high quality. The team wants to adopt DevOps to remove any friction in the process and accelerate the value delivery. This design science research proposes design artifacts for adopting DevOps in IAM by answering to the following research question:

How to adopt DevOps principles, practices and tools?

The sub questions below will help to build knowledge base to respond to the main research question.

- What is DevOps?
- What are the principles and practices of DevOps?
- What DevOps tools exist?
- What are the challenges in adopting DevOps?
- How to adopt DevOps in IAM?
- What are the challenges and success factors of adopting DevOps in IAM?

1.3 Summary

The end results of this research are three proposed artifacts for adopting DevOps in the IAM's service delivery; including the formation of the teams and the processes covering build, test and deployment of IDM system and onboarding new applications to IDM. The case study conducted in the Company X helped to identify the main problem areas in DevOps adoption, construct the design artifacts and evaluate how well they function. The evaluation highlights the challenges and success factors while adopting DevOps in IAM using proposed design artifacts, and present what are the special considerations that should be taken into account when adopting DevOps in this area. In addition, the conducted research shows that the organizations' operating model, supplier relationships, technology and tools, and mindset and skills have an impact on DevOps adoption in the teams.

This research begins with Chapter 2, which will introduce the reader to the research design including the research method, research process and data collection techniques.

Chapter 3 will explain the background of the main topic, before diving deeper into the subject of DevOps in Chapter 4, that will go through the concepts and the main constructs for the research. Chapter 5 will present how the case study was conducted and introduce the reader with the application domain. Chapter 6 goes through the analysis and design construction made in the case study, following with the evaluation of the results in Chapter 7. In Chapter 8 the thesis is finalized by presenting the conclusions, research limitations and suggestions for further research.

2 RESEARCH DESIGN

2.1 Research method: Design science

For centuries, natural science research has been focused on adding clarity and simplicity to complex objects and phenomena by describing their properties, behavior and interaction with one another. This descriptive type of research aims to simplify complexity and explore patterns in order to develop concepts and theories that help us to comprehend the world. However, most of our world today is composed of artificial objects and phenomena that are synthesized by human, and that are often linked to artificial functions and goals. (Simon 1996, 1–5.)

Design science is a prescriptive research approach based on a theory of design and action, which has been used as a basis of research for example in software engineering research, constructive research, prototyping and system development (Gregor 2006). Contrary to the natural sciences, the science of design is not concerned with how things are, but it focuses on researching how things should be, and how to design things in a way that artificial goals can be achieved. Design science research allows the researcher to look beyond the things as they are from an observing position and concentrate on how to improve things. Often, design science aims to solve 'wicked' organizational problems that require innovative and novel problem solving. (Hevner & Chatterjee 2010.)

Design science as a research method has raised interest especially around information technology. It can be argued that the prescriptive studies focusing on improving IT practices are of more value and interest than the descriptive studies that aim to explain and understand IT. One of the biggest reasons for this is the artificial nature of information systems and organizations that are built by human. (March & Smith 1995.) Another reason, as Hevner et al. (2004) claims, is that information technology research which is based on only descriptive research methods, is doomed to be reactive.

Design science methods have been used in information system research to build decision support systems, modeling tools, information system evaluation and change intervention methods, and information system governance strategies (Gregor & Hevner 2013). Aken (2004) argues that design science research method should be further utilized also in the management research, because *"understanding alone is not enough*". In addition to Aken (2004), also Hevner et al. (2004) suggest that by combining prescriptive research with traditional descriptive research, it could be possible to have a significant impact on organizations and improve the organizational performance.

DevOps is a framework that should always be adapted to its application environment (Erich et al. 2014). In this research, the focus is to create an effective design for adoption

of DevOps. The researcher will be an active participant in a practical case study: in addition to combining scientific knowledge base with knowledge from the application domain, the researcher aims to propose a solution to a real-life challenge. As this research will focus on solving a real organizational problem the design science is found to be the most effective approach.

Here, it is important to note the difference between developing a professional design and conducting a design science research. In the professional design, the designer only applies the existing professional experience and knowledge to find a solution to an organizational problem. As an example of professional design work is designing a software based on business requirements using existing, known best practices. In design science research, the knowledge base will be expanded to science, and the results should have a clear contribution to the existing scientific knowledge base. In addition, the results of the design science research will bring value to the stakeholder communities; other practitioners and researchers. (Hevner & Chatterjee 2010, 15–16.)

2.2 Research process

The design science research process is formed by the connections between the research, the research environment and the knowledge base of the research. These three aspects lay the foundation to the purpose of this research, which is to use information from the knowledge base (through theories and methods) to build an artifact that serves a specific purpose in the application domain (The case Company X and IAM) and evaluate how well the artifact suits for its purpose (Hevner et al. 2004; March & Smith 1995).

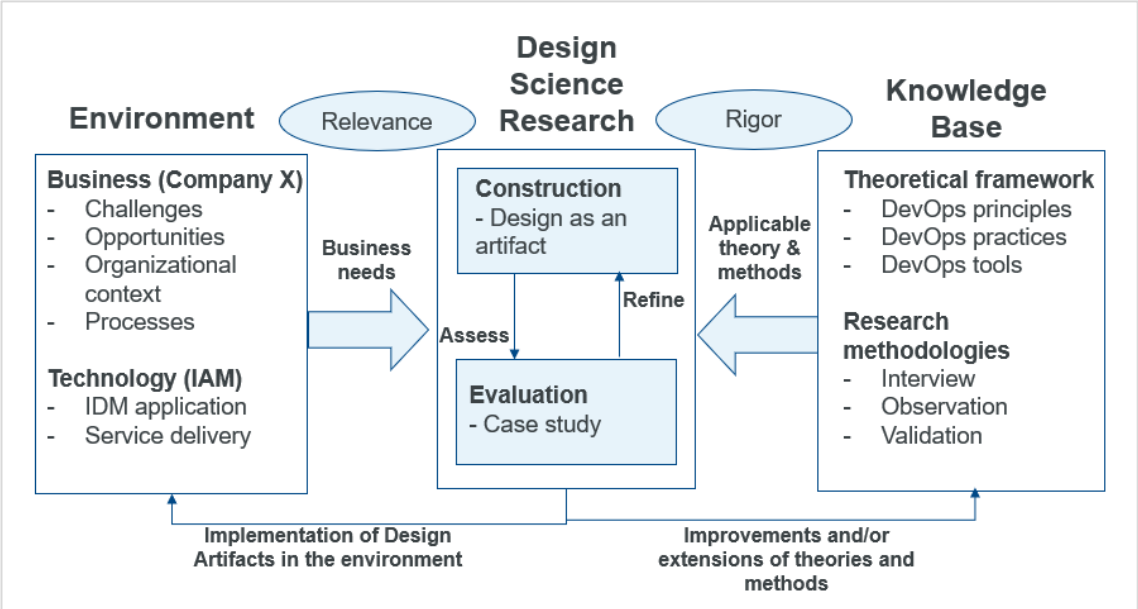

Figure 1 Design science research model (adopted from Hevner et al. 2004)

2.2.1 Environment

Before initiating the problem-solving process, the problem should be presented in its environment, where the search for the solution will be done (Simon 1996, 108). This design science research process begins with the inputs that connect the research project with its environment; Identity & Access Management in Company X, and the business problems related to delivering IAM services. The relevance refers to the requirements and the acceptance criteria collected from the research environment. The acceptance criteria determine whether the design artifact of the research project is appropriate or not. (Hevner & Chatterjee 2010.) Hevner et al. (2004) state that the fundamental objective of design science in the field on information systems is *"to develop technology -based solutions to important and relevant business problems"*. The main requirement of this research is to develop a solution for adopting DevOps in Identity and Access Management area, and the acceptance criteria is derived directly from the business drivers present in a real organizational context; such as the speed of delivering IDM and onboarding applications (further discussed in Chapter 5). One of the limitations in the design science method is the difficulty of completely understanding the environment in which the design artifact is built to operate; including the infrastructure, information systems, applications, communications and the architecture (Hevner et al. 2004; March & Smith 1995). This research seeks to address this limitation by conducting initial interviews and analysis, to gain deeper understanding of the environment.

2.2.2 Knowledge base

The research rigor refers to the phase in the design science research process in which the research is connected to the knowledge base that provides the foundation for the research (Hevner & Chatterjee 2010). While descriptive natural science research offers informing knowledge to prescriptive design science research, the design science contributes natural science research by providing novel artifacts. In addition to scientific knowledge, design science research also considers the role of the application environment and the participating people's cognitive and social capabilities in the research knowledge base. (Gregor & Hevner 2013.) The knowledge base in design science research is illustrated in a Figure 2, in which Ω refers to the descriptive knowledge and Λ to the prescriptive knowledge.

This research will use descriptive research as a knowledge base to draw relevant information to the research questions. From existing literature, it is possible to conceptualize what is DevOps in the first place and what do we already know about it. This research will combine the scientific knowledge with the knowledge from the application domain. The knowledge from the application domain includes the experience and expertise of the

people in the application domain, and the meta-artifacts such as existing processes and systems. Knowledge derived from the application domain guides towards better understanding of the environment and building of effective design artifacts.

Most importantly, the rigor ensures that the results of the research are not based on only existing best practices, but the knowledge base is extended with scientific research, which ensures both the innovativeness and scientific contribution. Eventually, rigorousness can be assessed by how applicable and generalizable the design artifacts are. (Hevner & Chatterjee 2010; Hevner et al. 2004.)

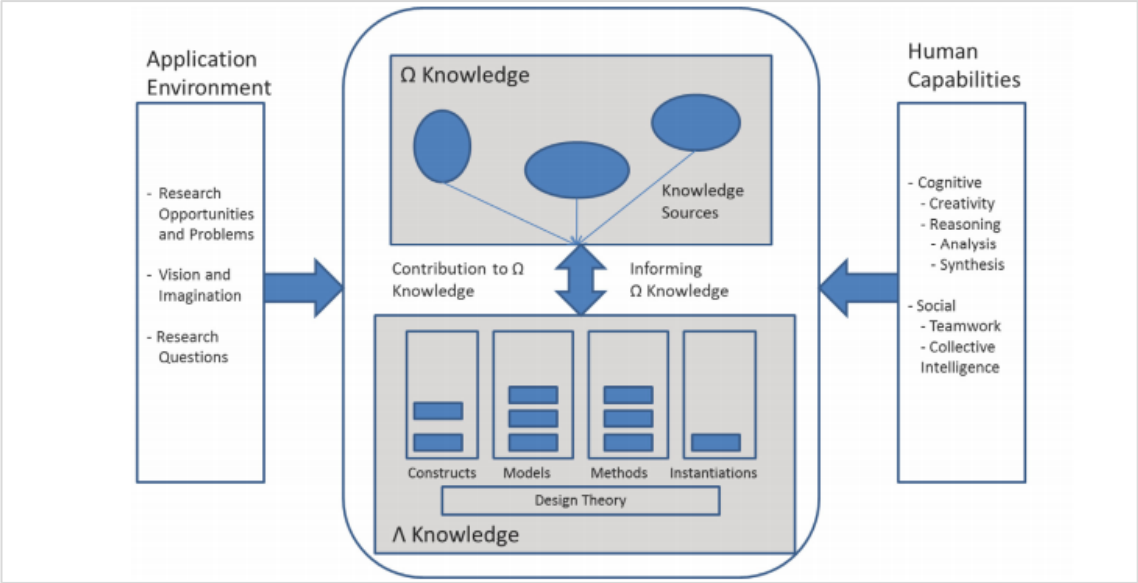

Figure 2 The Role of Knowledge in Design Science Research (Gregor & Hevner 2013, 344).

2.2.3 Design construction and evaluation

Based on Hevner et al. (2004) the critical nature for design science research approach in information systems research is to find the IT capabilities that could help information systems to expand into new areas. This research intends to construct artifacts that describe how to adopt DevOps in Identity & Access Management. The construction of design artifacts aims to propose *what could be/should be* by linking the as-is process or model to the new model proposition (Zimmerman & Forlizzi 2008). The whole design process comprises of the iterations where design artifacts and processes are built and evaluated, usually with a goal of improving performance. (Hevner & Chatterjee 2010; March & Smith 1995.)

After the design artifacts are built, they should be evaluated against the application environment. The evaluation in this research is done by conducting a case study where

artifacts for DevOps adoption are applied and analyzed in use (see Chapter 5 for more details on the case study process). Without evaluation of the artifact, the research is presenting only an assumption that the designed artifact might be useful for solving a problem or improving performance (Venable et al. 2012).

2.3 Data collection techniques

The data collection technique used to construct the knowledge base for this research is a combination of a literature review, interviews and observation/document review from the application domain.

2.3.1 Literature review

The literature for this research was collected by using Google Scholar database and the Volter database offered by the University of Turku. From Volter, the researcher has access to scientific databases of ACM – Association for Computing Machinery, IEEE, SAGE, ScienceDirect, SpringerLink, Wiley Online Library, among others. To assure the freshness and relevance of the material, most of the articles and books that were used as a knowledge base were not more than five years old. Following keywords were used to search for the articles: *"Devops", "Devops adoption", "Devops principles", "Devops tools", "Devops practices", "Continuous delivery", "Continuous deployment", "CI","CD", "Continuous feedback", "Agile", "Lean IT", "Lean Software"*. Also the keywords *"Devops identity and access management"*, *"Devops identity management*", *"Devops access management*" and *"Devops IAM"* were used to search previous or related research on this application domain but provided no useful results. In addition to the articles collected from these databases, books and publications from some of the most known DevOps practitioners and researchers such as Kim Gene, Jez Humble, John Willis and Gary Gruver, have been used to build up understanding of the concepts and practices around the topic. The overall list of references can be found from the Chapter 9.

2.3.2 Interviews

Qualitative in-depth interviews are often conducted individually, and they provide an opportunity to get more detailed data than through surveys (Boyce & Neale 2006). Therefore, this type of interview technique was chosen as the most suitable data collection method from the application domain, considering the novelty of the topic. The purpose

of the interviews was to gain understanding on what is the knowledge level of DevOps in the application domain, gather insights, and finally collect data to evaluate the design artifact.

Interviews can vary from highly structured to less structured, conversational interviews. In highly structured interviews, the researcher has pre-prepared the interview questions in advance, and the same questions are asked by all interviewees in the same way. Often, the research that focuses in building a model, uses more of lightweight, unstructured interviews, while in the research that aims to test an existing model, often structured interview techniques are more typical. (Wengraf 2001.) In this research, the aim is to develop a design, and semi-structured interview was chosen as an interview method. Semi-structured interviews allow both the interviewer and the participant to dive into deeper discussions whenever applicable, to collect richer data. There is a significant benefit of having a prepared set of questions, but also having the flexibility to skip a question or ask follow-up questions to adapt based on the interviewee's background and experience.

Qualitative interview process comprises of the following steps: 1. Preparing for the interview 2. Constructing the interview questions 3. Implementing the interview and 4. Interpreting the data (Turner 2010). In the interviews from the application domain, the first step is to define the scope of the interview and to list the relevant interview participants. The interview questions will be constructed so that they work as a guidance to the interview topic. Finally, the interviews are conducted by taking notes, which are later interpreted and analyzed by categorizing the responses thematically.

2.3.3 Observation and document review

The other sources of data include the data collected through observations and by reviewing documents from the application domain. The observation helps researcher to gain better understanding of the case (Stake 1995, 60). The observation will be done by participating in the team meetings, most of the project meetings and scrutinizing the ways of work in the case company's Identity and Access Management area. Collection and review of documents will be used as another source of data for the research study. Documentation on the organization charts, architectural materials, Identity and Access Management solution design documents, processes and internal presentations are reviewed to gain better understanding on the application domain and provide meta-artifacts for the design process.

3 BACKGROUND

Starting from the evolution of the role of information technology in the business, this chapter will discuss how the growing importance of technology has impacted business, especially in the area of software development, leading to the emergence of DevOps.

3.1 The changing role of IT in the era of digitalization

Back in the days, the role of information technology was mainly to support business processes. The IT function of the organizations was focused on delivering reliable, scalable and secure IT services. (Haffke et al. 2017, 102.) A good example of the software development from those days are the early enterprise resource planning (ERP) systems, that were built to support the business processes by integrating data from functional silos (Jacobs 2007). The large and stable enterprise information systems were optimized for operational cost efficiencies and therefore, the whole IT landscape in the companies remained rather static. Very few changes were made to the applications and they were done over long cycles. (Ravichandran et al. 2016; 16.)

Today's business environment can be characterized as more and more complex and fast-paced due to globalization and technological improvements. The fast speed of technological advancement is also impacting customer preferences; customers demand higher quality and better customer experience. As customers expect a constant change and engagement from the companies, old business models are challenged with new digital ways of doing business. (Ravichandran et al. 2016, 37; Virmani 2015; 78.)

Now, there is a shift towards companies seeing IT not only as a cost center, but as a business driver and enabler for digital transformation. Businesses have slowly come to realize that regardless of the industry, technology could be used to differentiate and acquire new customers or markets. (Haffke et al. 2017, 102; Horlach et al. 2017, 5420.) IT and business professionals see that continuous innovation and leanness are required to stay competitive. Based on a research by Freeform Dynamics (2015), more than half (54%) of IT and business professionals see that in addition to their core business, their companies have evolved into software companies. Some had emphasized that the software delivery should be seen as a core competence of a company.

Despite the evolution of IT from supporting role to a central role in the business, the design of the IT function has not changed much. Therefore, IT executives are struggling between both being innovative and responsive in the era of digital disruption, and still maintaining and developing the core IT capabilities (in other words; "keeping the lights on"). The organizations do recognize the need to introduce changes and re-design the IT organization, however the key challenge is how to build the organization in a way that it

enables IT to become innovative and exploratory, and still remain reliable and cost efficient. (Haffke et al. 2017, 103; Horlach et al. 2017, 5420.)

Bimodal IT is a concept of designing IT function that enables the organization to use IT to explore and innovate, but also to improve and maintain the existing IT services. It is based on the classification of enterprise applications according to the stability of the requirements they receive, and whether the application is evolutionary or experimental of nature (Sharma 2017, 121). Bimodal IT is defined in Gartner IT Glossary as *"the practice of managing two separate but coherent styles of work – one focused on predictability (Mode 1) and the other on exploration (Mode 2)*". This is illustrated in Figure 3.

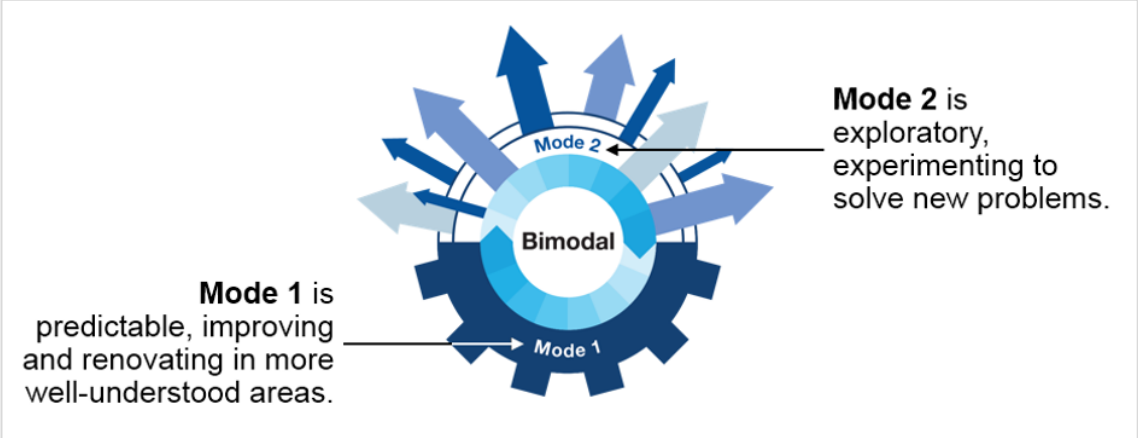

Figure 3 Bimodal IT (Gartner IT Glossary)

Haffke et al. (2017) found three main reasons why organizations adopt bimodal IT model:

1. The need for IT Agility: organizations establish bimodal IT model to be able to respond more flexibly to business requirements.

2. The need for explorative IT capabilities: organizations establish bimodal IT model to develop a "start-up" minded culture that encourages exploration and innovation.

3. The need for structural alignment with business: organizations establish bimodal IT model to align their structure to the contradictive business demands.

Sometimes Bimodal IT model is seen as an end state. However, this model leads organization into struggles as it divides the IT into two silos: the internal, stable and predictable backend silo, and the agile, flexible customer-centric silo. These two Modes should rather align how they operate as in the end they are both working towards the same business objectives. Often the "keep the business running" -applications and the legacy systems are the ones that accumulate most of the hidden waste, and therefore, affect the overall costs and improvement efforts negatively. But also, it is often on these bottleneck areas of IT where the investments eventually yield higher returns. (Sharma 2017, 93; Gruver et al. 2015.) Therefore, when formulating the overall digital strategy, the companies should specify whether Bimodal is the target model, or whether the Bimodal IT is a temporary

step, or a pilot structure, from rigid traditional IT organization towards unified, agile IT organization. (Holach et al. 2017, 5428.)

3.2 Traditional software development

This section will take the reader couple of steps back in time and briefly introduce the history of software development. The software development life cycle models have been created to structure the software development related tasks and activities. In the early years of software development, the main focus was on the requirements. All the way through the 1970s and 1980s, the software development life cycle models emphasized processes, strong control and structured development, which were achieved by following **sequential** software development models. (Kneuper 2017, 41–51.)

One of the first documented sequential software development model worth mentioning was developed for the SAGE project already in 1956. The SAGE project developed the Semi-Automated Ground Environment system for the air defense of the United States and Canada, and it is claimed to be the most ambitious information processing project of the decade. (Boehm 2006, 13.) Later, what became the best-known traditional software development model, is the sequential **waterfall model** (Balaji & Murugaiyan, 2012).

The waterfall model was first introduced in 1970 by Dr. Winston W. Royce in his famous article *Managing the development of large software systems*. In this very much referred article, Royce is providing his opinion of the waterfall approach, stating that he believes in the concept, but simultaneously he thinks that the implementation of it *"--is risky and invites failure".* The development based on a waterfall model is process-centric and thus characterized by strictly specified tasks, outcomes and roles. The main pattern is to start with collecting all requirements in the start of the project, classify them to functional and non-functional requirements, and then developing and testing the software linearly towards the production. (Ravichandran et al. 2016, 17.)

The main advantages of the waterfall model according to Balaji & Murugaiyan (2012) are the clear requirements, specified time periods for each phase, the linear and easy implementation, minimal amount of resources needed, and the documentation that follows the software quality. However, Nerur et al. (2005, 74) note that the large amount of documentation required eventually leads to formal and explicit communication. The water fall model, and the various documents required by the approach are illustrated in a Figure 4.

Stober & Hansmann (2010, 1–14) claim that the documentation and planning-oriented approach of the waterfall model present an illusion that the project is under control and everything will go well if the project plan and the original documented design is followed. To describe how the things will actually proceed in practice, they cited Murphy's law,

which states that anything that can go wrong, will go wrong. The world keeps changing, the requirements keep changing, and so does the design along with them. Thus, the authors conclude that *"Only a fool will blindly trust and follow a plan without questioning it every day."*

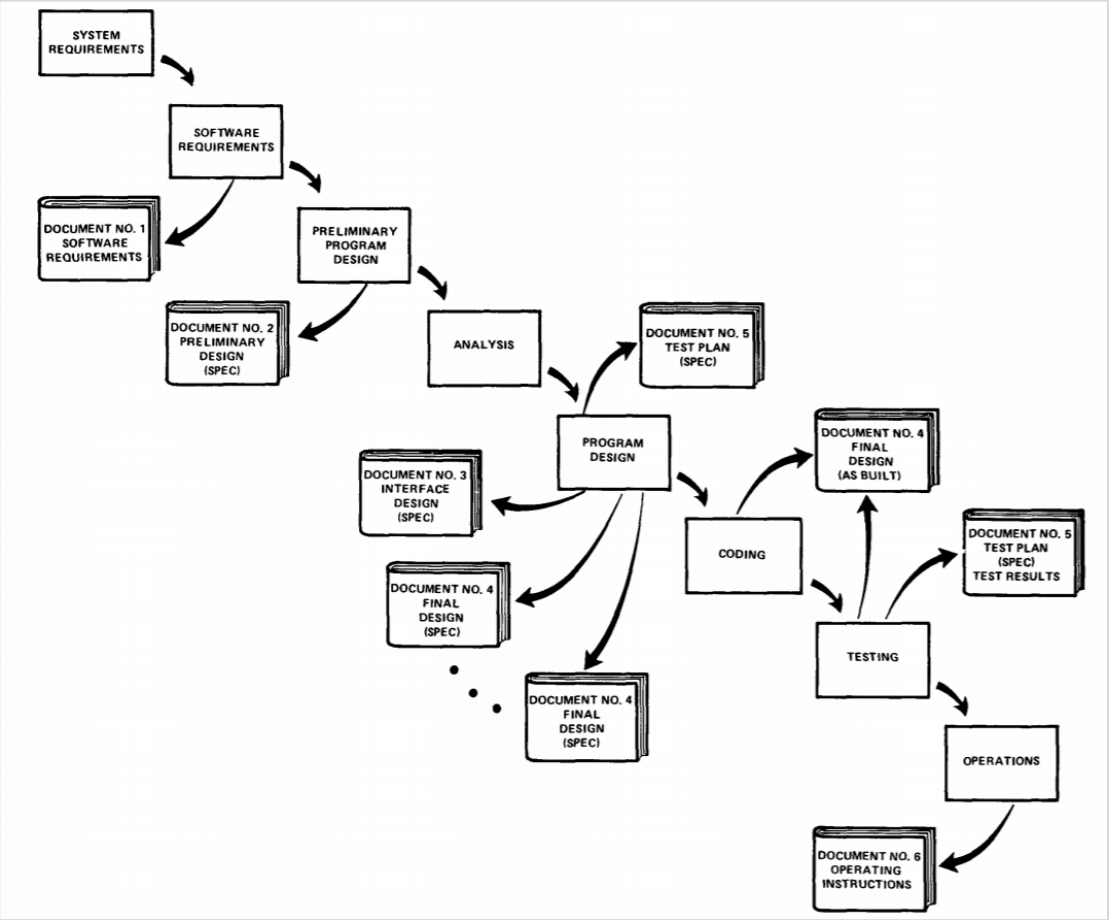

Figure 4 Waterfall model with six unique required documents. (Royce, 1970; 333)

Despite the risks raised by Royce back in 1970, the popularity of the sequential approach did not see a decline until 1990s. During 1990s more agile models started to slowly emerge and new methods, such as iterative development, were taken into use in the software development world. This shift was driven by the pressure from increased competition and the software time-to-market demands. (Kneuper 2017, 49.) The requirements that were previously collected in the start of the project, started to change constantly due to the speed of changes in the industry and technology. The software products became more user interactive, and the customers started to have difficulties to state their requirements beforehand. At the same time, the customers continued becoming more demanding of the software than ever. (Fitzgerald & Stol 2014; Boehm 2006, 18; Cohen et al. 2004, 3.) These changes in the business marked a critical change to the existing IT practices.

3.3 Agile software development

Agile software development, in its essence, is an iterative working style composed of different techniques and working methods that enable faster problem solving and quicker response to changing software requirements. Several different agile methods have been developed throughout the years such as Scrum, Extreme Programming (XP), Crystal Methods, Feature Driven Development and Dynamic Systems Development Methodology (DSDM). (Kaur & Jajoo 2015, 836; Stober & Hansmann 2010, 13; Cohen et al. 2004, 12.) Agile, as a term, was first coined in the Agile Manifesto which was signed in 2001 by independent thinkers who had been actively developing agile approaches to software development and project management during 1990s (Highsmith & Cockburn 2001, 6). The original manifesto combined the shared agile values and paved the way for further adoption of agile way of working. The following four core values are stated in The Agile Manifesto (Beck et al. 2001.):

1. Individuals and interactions over processes and tools.
2. Working software over comprehensive documentation.
3. Customer collaboration over contract negotiation.
4. Responding to change over following a plan.

The principles emphasize the importance of people in the software development process. The satisfaction of a customer, focus on the business value and constant feedback is at the heart of the agile methodology. The central idea is to work in short feature-driven iterations with small increments, welcome changes and requirements continuously in the process, and reduce the amount of detailed documentation. (Beck et al. 2001; Highsmith & Cockburn 2001; 6.)

The benefits from agile development are mostly related to the ability to respond to the constantly changing requirements. As the software updates are made in smaller and regular batches it is easier for businesses to adapt to new information. Also, the detection and correction of mistakes and false assumptions is easier and takes less time with agile. The faster detection of possible errors not only reduces related costs but also encourages faster feedback and organizational learning. In agile software development, the developers can apply their learning from feedback immediately and push software code into production faster and more often. The continuous design improvement enhances the software quality and performance. Agile teams are self-organized and managed through leadership and collaboration instead of traditional command-and-control. (Ravichandran et al. 2016, 18; Nerur et al. 2005, 74–75.) Table 1 summarizes the main differences between traditional software development and agile software development with regards to fundamental assumptions, control and management style, knowledge management, role assignment, communication, the customer's role in the development, project cycle, development model, and desired organizational structure and technology.

Table 1 Traditional versus agile software development (Nerur et al. 2005, 75)

	Traditional	Agile
Fundamental assumptions	Systems are fully specifiable, predictable, and can be built through meticulous and extensive planning.	High-quality, adaptive software can be developed by small teams using the principles of continuous design improvement and testing based on rapid feedback and change.
Control	Process centric	People centric
Management style	Command-and-control	Leadership-and-collaboration
Knowledge management	Explicit	Tacit
Role assignment	Individual – favors specialization	Self-organizing teams – encourages role interchangeability
Communication	Formal	Informal
Customer's role	Important	Critical
Project cycle	Guided by tasks or activities	Guided by product features
Development model	Life cycle model (Waterfall, Spiral, or some variation)	The evolutionary-delivery model
Desired organizational form/structure	Mechanistic (bureaucratic with high formalization)	Organic (flexible and participative encouraging cooperative social action)

Large-scale software development projects have multiple characteristics, such as projects with more than one team, a number of external consultants, extensive integration to other IT systems, and migration of large amount of data from legacy systems to the new one (Rolland 2016). Cao et al. (2004) emphasized that agile methodologies are not adaptable for this kind of complex large-scale projects, noting that especially architectural design should be made more upfront in the large-scale projects. Collaboration in large teams is also seen as one of the common obstacles, as people tend to break the big team into smaller sub-teams to better collaborate. Eventually, this habit leads to inconsistency between the sub-teams and possible re-work. (Elshamy & Elssamadisy 2007.) Balaji & Murugaiyan (2012) claim that agile development is more suitable for smaller projects, because in larger agile projects it is difficult to estimate the resources required for the project throughout the whole life cycle. The main challenge in scaling agile to larger development projects is working across knowledge boundaries that exist between various stakeholders of the development project. (Rolland 2016.)

However, other studies suggest that agile approach is applicable also for large projects. For example, Lindvall et al. (2004) conducted a pilot study in Motorola, which convinced the company that they can successfully use agile methods in the development of large, complex and safety-critical systems. After piloting agile methods in organizations such as Nokia, DaimlerChrysler and ABB, Lindvall et al. (2004) concluded that the agile methods are applicable also in the projects in large organizations. The biggest challenge was fitting the agile projects in the existing organizational environment which encourages traditional approach. Also, Kaur & Jajoo (2015, 832–836) studied the applicability of agile on large scale development projects, and as a result, they found that welcoming the changing requirements incrementally throughout the project was highly applicable in the large-scale development projects. To cope with the competition and complexity of the business environment in the 21st century, regardless of their size, companies today have started "scaling agile" and seeking for more adaptive ways of delivering and managing information technology efficiently (Kneuper 2017, 50–51 ; Ravichandran et al. 2016, 17–18; Nerur et al. 2005, 73).

3.4 Emergence of DevOps

Companies and IT teams that have adopted agile development approaches have begun to deliver IT services at such a fast pace that the operations teams are not able to keep up. Thus, IT teams are not capable of releasing new versions of the software or making changes to the software as fast as it would be required. (Sharma 2017, 5.) The reason in most companies is "the wall of confusion" between development and operations that makes collaboration, communication and problem solving between them very difficult (Artac et al. 2017). Often there is hardly any interaction between development teams and operations teams before the application is deployed in the production (Soni 2015). It has been recognized that the reasons for this are the different goals, mindsets and processes that are followed by development and operations teams. (Wettinger et al. 2016, 317.) In large organizations, the development and operations practitioners are also often situated apart from each other in the organizations' reporting structure and might not have ever even met (Sharma 2017, 149).

The "wall of confusion" has its roots in the core, chronic conflict (Figure 5) between the IT development and IT operations. IT operations usually have a goal of ensuring the availability of applications that are running in a stable and reliable environment, and the IT development aims to respond to the business requirements as fast as possible, delivering new features and solutions to the demanding customers. While the operations values

predictability and mechanistic working style, the development values creativity and ability to be agile in an uncertain environment (Sharma 2017, 149; Kim et al. 2016; Ravichandran et al. 2016, 28.)

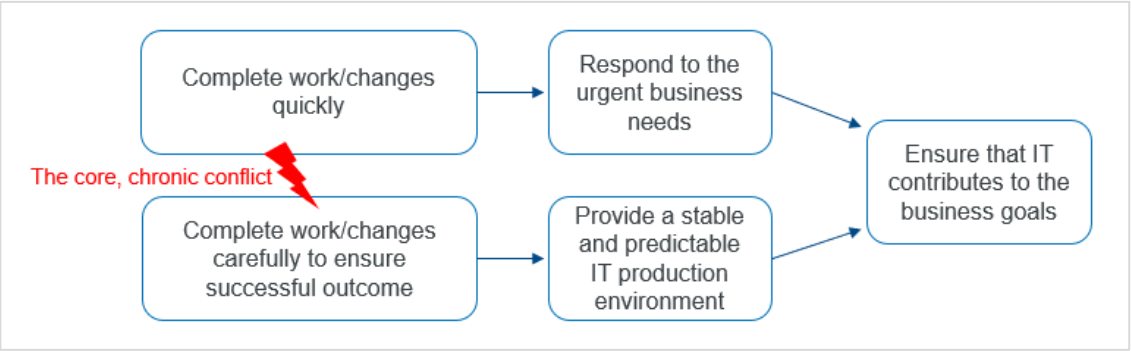

Figure 5 The core, chronic conflict adopted from Kim et al. (2016, 356)

DevOps movement emerged as a solution to the friction between Dev and Ops, extending the agile principles to cover not only development, but the whole software delivery process (Virmani 2015, 78; Kim et al. 2016). Recall from section 4.1. the bimodal IT model; at organizational level DevOps aims to build a bridge between the predictive 'keep the business running' IT (Mode 1) and the explorative and agile IT (Mode 2) by breaking down the development versus operations silos. The silos are broken down by combining the expertise and building smaller, efficient cross-functional teams that follow DevOps principles and practices (Wiedemann 2017; Balalaie et al. 2016; Ebert et al. 2016; Ravichandran et al. 2016, 6; Lwakatare et al. 2015). In addition, DevOps aims for faster, continuous delivery of value by reducing manual work and by building automated deployment pipelines (Verona 2016, 3–5).

Literature reviews on DevOps (Ghantous & Gill 2017; Erich et al. 2017) recognize that a single definition of DevOps or its meaning, is not agreed upon. From one perspective, the automation and continuous software delivery emerge as key themes, and DevOps is perceived as an automated chain of tools for agile software development. Technically speaking, DevOps is defined as a set of practices that aim to decrease the time between making a change and deploying it to the production, while ensuring high quality. (Bass et al. 2015; Soni 2015).

Another perspective defines DevOps as an organizational approach, instead of merely as a set of practices or tools. The concepts of communication and collaboration are frequently discussed elements of DevOps, highlighting the change that DevOps brings to the work culture and processes. Sharma (2017) sees DevOps as a holistic movement that puts the cultural transformation across functional silos above the processes and tools. Dyck et al. (2015) provide a comprehensive definition of DevOps as *"--an organizational approach that stresses empathy and cross-functional collaboration within and between*

teams – especially development and IT operations – in software development organizations, in order to operate resilient systems and accelerate delivery of changes." Altogether six different perspectives on what DevOps is has been identified (Erich et al. 2017):

1. Merging of development and operations.
2. Development that supports operations.
3. Centralized group with two aims: the first aim is to achieve visibility, consistency, efficiency and predictability in development and operations. The second aim is to support the transition to agile methods.
4. Incentive alignment among every party involved in the software development and operations.
5. Cultural movement.
6. Area of expertise combining the development and operations skills.

Many studies have been conducted to learn what are the benefits of DevOps (Ghantous & Gill 2017). The DevOps pioneers Jez Humble and Gene Kim collected evidence of the concrete business value from DevOps adoption between 2013 and 2016 resulting with data from more than 25 000 practitioners. (Kim et al. 2016.) The data showed that DevOps adoption had resulted in improvements in multiple different areas, such as throughput and reliability metrics, information security, lead time and amount of deployments, mean time to recover, productivity, profitability, market share, and employee satisfaction.

Another supporting evidence is a smaller study conducted by Freeform Dynamics in 2015, which surveyed IT and business professionals on DevOps and found out that majority (~70%) of them had already adopted DevOps to some extent in their organization. The comparison between the advanced DevOps adopters and the early and non-adopters showed that the advanced DevOps adopters had gained both competitive market performance -related benefits and business scorecard benefits. The advanced DevOps adopters saw improvements in their market performance by the ability to act quicker on the market, open new markets, and create completely new business models. When it comes to the financials, the advanced DevOps adopters had improved their overall revenue and profit.

4 THEORETICAL FRAMEWORK

This chapter is built upon existing literature and previous research and provides the theoretical framework for this research. The principles, practices and tools presented here are used to later formulate the design artifacts for adopting DevOps in Identity and Access Management.

4.1 DevOps principles

DevOps is built upon three main principles that have been influenced by other existing philosophies. The author and researcher Gene Kim defined the three principles of DevOps, referring to them as **The Three Ways**.

4.1.1 The First Way: Optimize the work flow

The First Way establishes the foundation for DevOps by emphasizing that the end-to-end performance of the whole software delivery process matters. Software delivery process has often been compared with manufacturing, and earlier it was not new that the IT departments were adopting the practices from manufacturing to optimize the processes. Typically, the mass manufacturers produced standardized products in large volumes to optimize, and the production line workers were specialized to execute tasks as efficiently as possible. Also, in traditional IT, the trend has been to implement standardized, cost-efficient software. The work is thus organized into specific tasks that require specialization; the developers write code, testers test the code, and the operations person is responsible for deploying the software. However, there is a significant difference between software delivery and manufacturing because unlike mass manufacturing, software development is a highly creative process that requires collaboration and creativity to deliver value. (Hering 2018.)

Strict division of work between the development and operations causes friction to the work flow (Kim et al. 2016, 15; Ravichandran et al. 2016, 34). Another difference to note when comparing software development process to manufacturing is that in software delivery the work items are non-tangible, while in manufacturing, the work flow from preparing raw material towards the final product is highly visible (Hering 2018, Sedano et al. 2017, 130; Ravichandran et al. 2016, 35; Kim et al. 2016, 15–16). Often in software delivery, the work is invisible to the developer, and the line between a work item and a final product is blurry, especially in the deployment phase. The non-tangible nature of the

work makes detection of the friction points difficult and therefore, the good visibility over work becomes crucial in software delivery.

DevOps aims to optimize the software delivery and remove friction by borrowing thoughts from modern manufacturing philosophies, such as from the Lean. The Lean paradigm emerged to renew the legacy manufacturing by providing methods to manufacture with "*half the human effort in the factory, half the manufacturing space, half the investment in tools, half the engineering hours to develop a new product in half the time".* The first Lean practices were created and experimented in Toyota factories in 1950s by Taiichi Ohno, who came up with the concept of waste, or "*muda"*, in Japanese. (Womack et al. 1990.) In Lean thinking, waste means any non-value-adding activities, variations or overburden that appear in the work flow (Ikonen et al. 2010). The Lean emphasizes visibility to the work, automation where possible and employing teams with a rich skillset. By doing so, it aims for a faster delivery of high-quality products with reduced costs, by removing any waste that appears in the processes. (Humble & Farley 2010, 16.)

To help recognize where non-value adding work is piled up in the work flow in IT, Poppendieck & Poppendieck (2007) apply Lean manufacturing methods by translating the seven categories of waste to the software development. The Table 2 summarizes inputs from Poppendieck & Poppendieck (2007), Kim et al. (2016) and Ravichandran et al. (2017), providing categorization of the wastes in software development with examples of possible business outcomes. Note, that the wastes are often interrelated, for example defects often lead to extra work when the coding must be repeated to fix the bug. Another example is that delays lead to work in waiting and the re-learning can be seen as extra work, as well. However, majority of the waste results from the friction between people and teams, and the manual tasks executed by them. The detection of the potential development points in the software delivery flow can be started by conducting a value stream mapping exercise, with the purpose of identifying and drawing the current processes to identify the friction points where the processes could be improved (Hering 2018; Sharma 2017, 49; Fitzgerald & Stol 2014).

Further on, different agile techniques can be used in DevOps teams to track the progress of the work and collaborate more effectively. Instead of having heavy documentation and plans, the goal should be to do continuous planning. The handoffs should be made as seamless as possible, so that the teams are able to collaborate effectively across functional boundaries (Sharma 2017, 28). The work flows can be visualized for example by using Kanban boards. Kanban works as a control mechanism that helps to measure the lead times of individual work items and limit the work in progress (Ikonen et al. 2010).

Table 2 The 7 Wastes in Software Development (adopted from Ravichandran et al. 2017, 35–36; Kim et al. 2016, 24–25; Poppendieck & Poppendieck 2007)

Category of Waste	Examples	Possible business outcome
Work in waiting	Requirements that are not in development, unsynchronized code; code that is not merged to the master code, untested/undeployed code	Increased capital and operational costs
Extra Work	Delivering features customers do not need or want, procuring extra capacity due to unanticipated performance requirements, documentation that is not used, reviews and approvals that do not add value to the output	Delays, cost over-runs, and budget problems
Relearning	Same people learning the things, insufficient knowledge sharing, poor collaboration	Inefficiency
Handoffs	Development/QA handoffs	Application launch delays; increased cycle times
Motion	Developers constantly switching	Lost productivity; talent erosion
Delays	Excessive release backlogs and bottlenecks, infrastructure and data not available for testing, change reviews; security and compliance audits	Slow time-to-market and value; lost opportunities
Defects	Badly designed and poor-quality code, non-functional performance issues	Lost customers and revenue; negative brand impact

4.1.2 The Second Way: Create efficient feedback loops

The Second Way is about creating efficient feedback loops that enable fast validation and detection of errors or anomalies. The fast feedback and feedforward are vital mechanisms to avoid problems moving forward downstream and starting of new work that builds upon existing errors and generates more of a technical debt. (Kim et al. 2016, 31) Essential about creating efficient feedback loops to the software delivery process is the shift left testing approach. In shift left testing the tests are executed as early as possible in the delivery, which means that also the defects are detected early in the process (Sharma 2017, 142).

For example, in waterfall -based software development, the software moves to testing only when all the design and development work has been done. At this point many problems may arise that could have been noticed earlier during the development if the testing would have been done earlier. Similarly, without having the efficient feedback loop in place the developers may use their time to develop a feature that eventually has no business impact or even makes the user experience worse. The faster the feedback flows from the customer back to the development, the faster the development knows whether to pull off the feature or continue to work on it.

Efficient feedback also means that the quality controls should not be embedded into heavy approval processes. The DevOps teams should be able to move quickly instead of being hindered by organizational barriers. (Gruver 2016, 60–61.) According to the State of DevOps report from 2014, the external approval processes for production deployments decrease IT performance, in terms of both throughput and stability. On the contrary, the approval free process and peer reviews correlate with higher IT performance (Forsgren 2018). Searching and solving problems should be made part of the everyday work of the person who can actually make a difference, and not relied upon a party that is distant from the work (Gruver 2016, 60–61; Kim 2016, 33).

Having feedback mechanisms in place for development and operations separately is not enough; creating efficient feedback means that the feedback flows between these two teams, but also between the other parties such as information security and service managers, so that they can have visibility over the work to the extent that is needed. (Kim 2016, 198–204.) The efficient and fact-driven feedback mechanisms also support the third principle of DevOps – the culture of experimenting and learning.

4.1.3 The Third Way: Enforce culture of experimenting and learning

The Third Way defines the importance of a culture that fosters continuous experimenting and learning. In complex environments, it is necessary to be able to detect and solve problems, learn from them, inject the learnings into the daily work and expand the knowledge in the whole organization. (Kim et al. 2016, 272.) Sharma (2017, 33) recognizes three improvement areas of a learning organization: the application, the environment and the process. All these areas should be under continuous experimenting and learning.

The famous Conway's law from 1968 states that organizations are constrained to produce designs that are copies of their communication structures. Thus, the communication culture of organization is doomed to affect the way the applications and processes are designed. The work environment in lower performing organizations is characterized with low-trust and 'command-and-control' type of management. Especially in many large or-

ganizations, the culture does not foster innovation. The main characteristics of large organizations are often strict governance, and key performance indicators that encourage people to do only the job required in their function or role. (Sharma 2017, 258.) In these low-trust cultures the first thing to do when something goes wrong is to find out who did it and punish them from it. This may lead to situations where for example developers will disclose information from the operations, or operations disclose information from the service managers to avoid being shown in bad light. Moreover, business can expect less and less innovative solutions as the people that are supposed to come up with them are afraid to fail. (State of DevOps Report 2014.)

On the contrary, DevOps culture is about building high trust and respect, having empathy and enforcing open communication channels (Ravichandran et al. 2016; 28–29). As discussed in earlier principles, DevOps aims to reduce the time to deliver by creating efficient feedback loops. In fact, also trust may be used as one of the key metrics to measure how fast the organization is able to deliver value. The Senior IT Specialist Lee Reid from IBM has presented that the speed to value depends upon trust (Reid 2015, June 22nd). The level of trust directly affects the time people spend on handing off work in the organization; the lower the trust, the less likely a person is to hand off anything forward to another person. Similarly, as people have higher trust among each other, the easier and faster they hand off work forward and backward.

In addition, the DevOps culture does not judge failures, because through failures there is a chance to learn and improve (Sharma 2017, 327; Kim et al. 2016, 273). The amount of bugs found in the software testing is not necessarily perceived as a sign of bad software quality; rather contrary. High rate of bugs found during the testing means that the testing is performing well and the team has avoided having bugs deployed into the production. Moreover, each member of the team is given end-to-end responsibility of their own work. Acquiring new skills and ability to collaborate across the functional boundaries are qualities needed in the experimental learning culture (Ebert et al. 2016). The teams should not be afraid of their work to be scrutinized by the colleagues or customers, because there should be nothing to hide. The performance of the work is everyone's shared responsibility and the problems are acknowledged and confronted by everyone in the team. The confidence and transparency to the work also eventually impacts how customers' see the company and its services and may lead to higher customer trust and loyalty (Kim et al. 2016, 206–207).

4.2 DevOps practices

While agile software development practices introduce continuous planning and adaptation to fast changing customer requirements, the DevOps practices can be used to extend

the continuous flow of activities all the way from the customer to the development, operations and back while all the time building the quality in (Fitzgerald & Stol 2014).

In many organizations the software delivery process is manual, meaning each step of the process is separate and possibly performed by a different individual. However, manual work is unpredictable, unrepeatable and error-prone. Having developers' resources stuck with manual deployment, fixing and testing reduces the time they could spend for more creative and value adding work. DevOps practices are enforcing the DevOps principles by building **a deployment pipeline**. A deployment pipeline consists of Continuous Integration and Continuous Delivery pipelines, and it can be defined as *an automated implementation of an application's build, test, deploy and release processes.* Although the implementation of the pipeline may differ, it is ultimately governed by the DevOps principles making the end-to-end process from development to operations automated and visible to everyone, improving the feedback and aiding the collaboration. (Humble & Farley 2010, 3–4.)

4.2.1 Continuous Integration

As a concept, **Continuous integration (CI)** is not new to the software development. According to Beller et al. (2017), CI was originally described by Microsoft in 1995, and became later known as a major software development practice in Extreme Programming (XP) approach back in 1999. In XP, the Continuous Integration practice has been described as *"New code is integrated with the current system after no more than a few hours. When integrating, the system is built from scratch and all tests must pass or the changes are discarded"* (Beck 1999). In DevOps, the element of automation is added to the CI. Here, Continuous Integration is referred to as "*The practice of triggering automatic process of building a deployment package*, *including code compilation, running unit tests, validating the code and assembling the code after each check in of a new piece of code or a change to the version control"* (Fitzgerald & Stol 2014). Figure 6 illustrates the activities in the CI pipeline.

The prerequisites for having Continuous Integration pipeline in place are the discipline to use version control and automation of the unit tests and builds. The whole DevOps team must be consistent with the use of version control; every tiniest change should be recorded there. (Humble & Farley 2010; 57). The main assumption with CI is that the application is broken until you prove otherwise. The code checked into the version control may have typos or errors. Thus, it is important to design the build automation in a way that the errors in code are mitigated and detected. (Sharma 2017, 298.). Without having automated unit tests ran in the CI pipeline the team can never be confident that the committed changes actually work. Unit tests allow quick testing of the behavior of the small

code changes in isolation from the other components of the software. (Meyer 2014; Humble & Farley 2010, 60.) The unit tests are written in practice by first defining the test methods, tagging the test methods with annotations in the source code, and marking the code as tests (Verona 2016, 86).

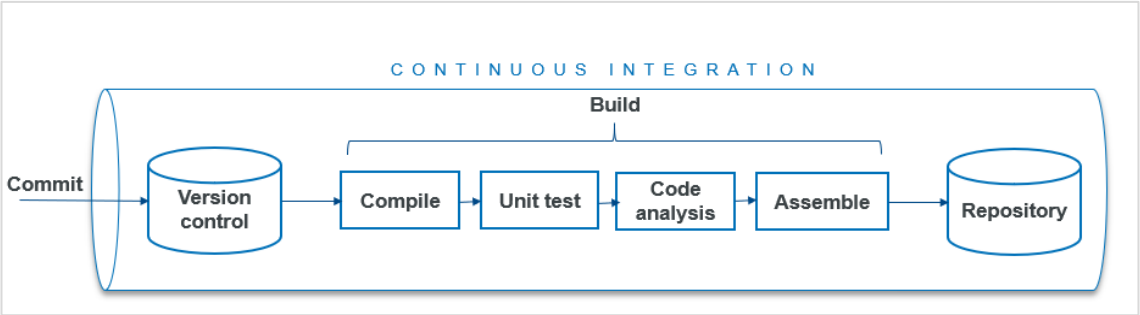

Figure 6 Continuous Integration pipeline

After the unit tests the static code analysis can be done to validate the code (Verona 2016, 61; Gruver & Mouser 2015, 83). The static code quality analysis provides information on the health of the code; defects, design issues, or if there are any vulnerabilities found in the third-party libraries. Validating the code in the CI pipeline will ensure that bad quality code is not handed off to the deployment. Finally, the code will be assembled into a deployable package that will be published to an artifact repository. Sometimes the deployment packages are published in the same version control repository as the source code and configuration files, but it is recommended to use separate pass-through artifact repository instead, wherein the artifact will be published when the build is passed on to deployment. (Sharma 2017, 137; Humble & Farley 2010, 165.)

4.2.2 Continuous Delivery

Continuous Delivery (CD) extends the CI pipeline to automatically deploy the healthy builds to production-a-like test environments where functional tests, integration tests, performance tests and security tests can be made (Sharma 2017, 16–18). The CD pipeline will be triggered from every new build artifact published in the repository. Ideally, the CD pipeline also automatically deploys the work finally to the production environment if all the tests are passed. In case any test fails in the pipeline, the deployment will stop, and the team will be informed that something is wrong about this build (Forsgren et al. 2018).

Practicing Continuous Delivery radically changes the release management process around the application. CD pipeline reduces the complexity around release processes by decreasing delivery risks and increasing the speed of delivery and feedback (Hering 2018). The changes that are delivered to production are as small as possible, thoroughly

tested, and they are deployed as often and as automatically as possible. However, the technical implementation of automated CD is not easy. For example, achieving the level of test automation that allows the team to trust that the deployment is safe, may take time. Therefore, in some cases human intervention is necessary at the final stage where the change is deployed to production. (Verona 2016, 16–17.)

Ebert et al. (2016) note that not all systems support automated delivery due to the environment constraints, for example safety-critical systems or systems in highly regulated industries. London Multi-Asset Exchange (LMAX) is a company that is taking advanced use of Continuous Integration and Continuous Delivery pipelines, but reportedly deployments to production are made only every two weeks outside of normal trading hours (Bird 2016).

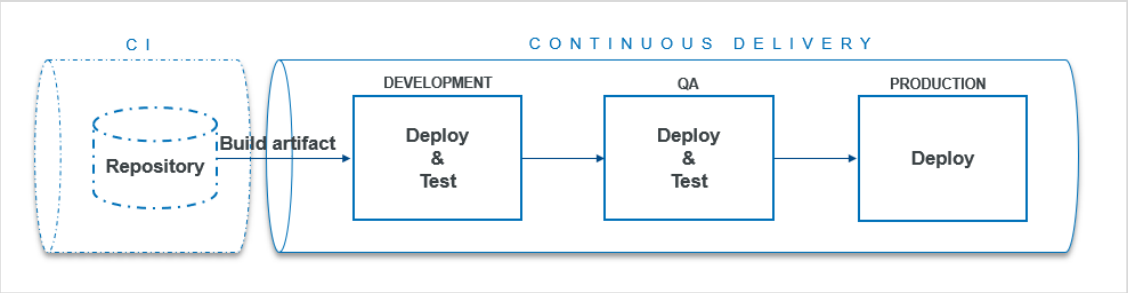

Figure 7 Continuous Delivery pipeline

Therefore, it is important to recognize that the terms Continuous Delivery and Continuous Deployment are often used interchangeably, although there is a slight difference between the meanings behind. In *the DevOps Handbook* published in 2016, Humble clarifies that also the 'push the button' a like deployments can be seen as Continuous Delivery as long as the deployment procedure can be made on-demand from the code base that is always at releasable state with a fast feedback loop that follows the deployment. The term Continuous Deployment, then, can be used to refer to the automatic, regular process of deploying healthy builds to the production. Hering (2018) notes that instead of actually deploying automatically into production, more important is the ability to do that. Citing Sharma (2017, 19), we can conclude that while Continuous Delivery is a *must*, the Continuous Deployment is *an option*.

4.2.3 Continuous Feedback

As presented earlier, creating efficient feedback loops is one of the main principles of DevOps. Making the feedback continuous from development to operations and all the way to the customer and back is prerequisite for success. The CI/CD practices are almost

meaningless without Continuous Feedback (Sharma 2017, 23). Several mechanisms exist to enable the Continuous Feedback loops in practice.

The first mechanism is to break the work down to smaller items so that the developers can validate continuously whether the work they deliver at the end of the iteration meets the requirements. Small development items are faster to test and as a result, the developer can get faster feedback on the quality of the work. (Forsgren et al. 2018; Sharma 2017, 242; Fitzgerald & Stol 2014.) The feedback from the work can be made even more effective when it is visualized; as an example, LED lamps can be used at the office to blink red light to the team when the build has failed (Verona 2016, 79). In virtual work environments, the team must be more creative and take some alerts or similar techniques into use.

Secondly, to enable continuous feedback from testing, automation must be in place. All tests including unit tests, functional tests, performance tests, integration tests, security tests and user acceptance tests must be ideally automated, in the deployment pipeline. (Sharma 2017, 23–25.) Incorporating the incremental agile development practices together with automated testing enables the teams to fail fast, learn quicker and develop solutions with built-in quality.

Thirdly, continuous feedback can be enforced by designing the systems to create telemetry for monitoring, issue resolving and further decision-making purposes. Telemetry is created by collecting the logs from the deployment pipeline and from each layer of the application stack; environments, application and the business logic. All this data should be then stored and transformed into metrics that can be visualized and monitored. (Kim et al. 2016; 198–203.) Two of the most important metrics when it comes to monitoring the system performance are latency and throughput. Latency refers to how long the system takes time to respond, in other words, how fast is the user experience. Throughput refers to how many requests the systems can respond in a specified time period. (Gonzalez 2017, 359–360.) Also, security needs to be included to the feedback loop to detect and respond to any risks during the software delivery. Instead of only annual security or compliance reviews, the security can be integrated to the deployment pipeline (Fitzgerald & Stol 2014). This approach to security is also referred to as ‘shift left security’, meaning that the security reviews are moved closer to the development phase where the vulnerabilities can be found and fixed quicker (Bird 2016, 19–26).

Fourthly, often in the case of outages, the operations team is pressured to solve issues, but they do not have visibility over what has been changed. This makes problem solving very hard, or nearly impossible to the operations team. (Kim et al. 2016, 196.) Continuous collaboration and sharing mutual understanding of the problems and improvement opportunities between development and operations is important mechanism for feedback. The operations can provide development feedback regarding performance or scalability issues

to help development to cope with defects and avoid issues with production load. Similarly, the feedback from the development guides the operations for example with service level requirements. (Ravichandran et al. 2016, 115–116.) The collaboration can be promoted by using a pull request among the team members every time someone pushes their changes to the repository. Pull requests are a compulsory activity in many large companies that are advanced with DevOps, such as Google, Facebook, Amazon and Etsy (Bird 2016). Essentially, pull requests are peer reviews that provide developers a possibility to gather feedback from another pair of eyes before the changes are merged to the main code. (Kim et al. 2016, 249–251.)

Finally, Sharma (2017) emphasizes that continuous improvement requires standardization of tools and practices. Without standardization the reusability of the tools, people and processes becomes more difficult and it becomes impossible to improve the processes or working styles. For example, in Google all the teams are forced to use the one specific tool for version control, to enable collaboration and the ability for anyone to see anyone's code (Gruver 2016, 62). Having at least few standardized processes and toolsets the organizations can become agile and make seamless transitions between practices, tools or people when needed.

4.3 Enablers and tools

Different tools and enablers can be used to enforce DevOps practices. In DevOps pipeline the goal is that all tools needed to deliver the application are integrated in a way that they form an automated chain of tools. The CI/CD tools are often dependent on the technology stack, but it is possible to standardize the DevOps toolchain for certain technologies such as Java, .NET or COBOL. (Sharma 2017, 122–123.) Here will be presented findings of the most common enablers and tools that can be used for managing and executing the work from towards Continuous Integration to Continuous Deployment and Continuous Feedback. There are hundreds of commercial and open source tools available that are developing new functionalities at the time of writing. Therefore, the ones presented here are provided only as an example.

4.3.1 Agile

Agile is a key enabler for DevOps. Scrum is an example of an agile project management methodology that enables the DevOps practices and helps the teams to collaborate, learn and improve. Part of Scrum are the frequent, daily "stand-up calls" that last only 10 to 15 minutes and bring everyone together to give updates and align their work. In this way,

the work is made more visible and planning is done continuously along the way. (Gruver 2016, 46–48.) For example, the team may prioritize the backlog items, raise concerns over something in the work or align their separate work streams in the brief daily meetings. In addition, the agile methods can be taken to use to capture the improvement points from the retrospective meetings held at the end of each cycle, or 'sprint'. These agile mechanisms enforce the culture of continuous feedback, learning and improving. (Verona 2016, 7.)

4.3.2 Version control

Version control tools (or Source Code Management (SCM) tools) allow storing, retrieving, identifying and modifying all the development project artifacts and relationships between them. Distributed version control allows team members to collaborate parallelly regardless of their geographic location, or regardless of the feature they are working on. The coherent use of version control ensures that every tiniest change to the code is recorded to the history logs of the code repository. Thus, if something gets broken, it is easy to roll back to the earlier version of the code. (Humble & Farley 2010, 32.) When using version control it is important to agree upon a branching strategy among the team. The branching strategy defines the common rules for how development branches are created from the master code repository, and what is the naming convention for the branches. (Verona 2016, 43–44.) Based on the State of DevOps report (2018), the organizations that used version control correlated with higher IT performance when it comes to deployment frequency, time to recover and the lead time for changes. Examples of common version control tools are Git, Bazaar, Mercurial, Source Forge, Subversion, Perforce, Gogs and Bitbucket.

4.3.3 Configuration management

Creating environments for development, testing and production is usually one of the slowest and documented manual activities in the software delivery (Gruver 2016, 30). By versioning also the configuration information enables **Infrastructure as a code**; re-production of any environment from operating system all the way to network configuration and software stack (Humble & Farley 2010, 32). Keeping configuration information in an abstract form is important in achieving speed of DevOps; the configuration files should refer to variables instead of concrete values, to be easily replaced (Hering 2018). By using common templates, or 'recipes', to reproduce configurations across servers minimizes the

differences between development, test and production environments (Bird 2016). Infrastructure Lead at Prezi stated that use of configuration management tool provides Prezi with agility: *"...If we want to start developing a new system tomorrow, by Noon we can put every infrastructure piece in place and be ready to go*" (Chef Blog 2013, January 22nd). Examples of tools for configuration management are Puppet, Chef, SaltStack and Ansible.

4.3.4 Cloud and containerization

Cloud is another key enabler for DevOps. Hosting resources in the cloud gives flexibility to scale on-demand as it eliminates the time used for waiting for hardware to be provisioned and reduce the cost of setting up data centers (Bird 2016; Soni 2015; Cukier 2013; Borgenholt et al. 2013) Cloud based infrastructure also increases availability (Sharma 2017, 146). Cloud native applications leverage containers to achieve further speed and consistency into the management of the development, test and production environments (Sharma 2017, 248; Ebert et al. 2016). Containerization means packaging an application and its configuration code from version control into a standardized image that is portable from one environment to another. Deploying DevOps tools in these containers makes it possible to reuse them in a service-like manner. Alternatively, container images can be used when publishing builds. The build artifacts can be packaged into isolated container images and further deployed to the environments. (Sharma 2017, 238–239.) Applications that package their services into these smaller independently deployable services are leveraging the cloud-native microservices architecture to enable Continuous Delivery (Balalaie et al. 2016). Examples of containerization tools: Docker, Kubernetes, Mesos.

4.3.5 Build tools

Build tools automatically create the build by compiling the code and packaging it for deployment (Machiraju & Gaurav 2018, 6). The build tools can be categorized into task-oriented build tools and product-oriented build tools. The focus of the task-oriented build tools is to reach the defined goal by executing a set of tasks. Product oriented build tools focus on the sequence of outcomes required to reach a goal keeping information of the state of the files in between the tasks. (Humble & Farley 2010; 145–146.) The decision of which build tool to use is usually made based on the type of product; complexity of the system and the system's code base (Verona 2016, 62). Examples of build tools: Ant, Maven, Boot, Gradle, Grunt and Make.

4.3.6 CI/CD tools

Continuous Integration tools help to integrate the builds as early and as frequently as possible (Ebert et al. 2016). CI tool is a build server that triggers the automatic creation of the build followed with automatic unit tests as soon as new changes are checked into the version control. CD tools are deployment servers that trigger the automatic deployment of the build to the test/QA/staging/production environment from each build artifact. (Verona 2016, 69–70.) New CI/CD tools are all the time appearing on the market with a purpose of merging all the tools needed to enable DevOps into one platform and provide possibility to easily integrate with the other tools.
Examples of CI tools: Buildbot, Hudson, TeamCity and TravisCI
Examples of CD tools: Serena Release and Octopus Deploy.
Examples of CI/CD tools: Azure DevOps, Bamboo, AWS CodePipeline and Jenkins.

4.3.7 Test automation tools

Test automation enables fast feedback and provides possibility to test more and faster, run test scenarios that are difficult to run manually and test with better consistency. Test automation tools should be integrated to the CI/CD pipeline to enable early, fast and Continuous Feedback. Any forms of tests can be automated from unit, integration and functional tests all the way to performance tests and load tests (Sharma 2017, 145). Also, the use of artificial intelligence in the test automation is growing and in 2018 Gartner gave a strategic assumption that by 2021, 30% of the application development organizations would be using AI in their test automation to enable DevOps (Gartner: Magic Quadrant for Software Test Automation 2018). Examples of test automation tools: Blueprism, Robot Framework, Selenium, Windmill, Protractor, Watir, Apache JMeter, Junit, Jasmine, Mocha and Cucumber.

4.3.8 Security tools

Security is challenged by DevOps when it comes to speed, minimizing waste, and building upon cloud-based infrastructure (Bird 2016). Security and compliance are often seen as a bottleneck as the regulations, rules and reviews add additional approval work to the release and deployment flow. Usually, there are not enough security professionals to be embedded to every DevOps team, but many tools can be used to integrate information security to the daily work of developers and operation engineers. (Ravichandran et al. 2016, 129–131.) A research by Forsgren et al. (2018) showed that integrating security to

the deployment pipeline leads to higher performance. Especially in the development of a mission critical software, some special tools should be integrated to the DevOps toolchain to execute compliance related tasks **(**Gall and Pigni 2018). As part of testing, we may implement following automated security tools to scan and analyze the code that is being built (Hsu 2018, 39; Kim et al. 2016, 318);

- Static Application Security Testing (SAST) tools scan the source code or the binary code and notices flaws and defects. Can be implemented in the developers' integrated development environment (IDE) or as part of the build. Examples: Veracode, SonarQube, FindSecBugs, Fortify, Coverity
- Dynamic Application Security Testing (DAST) tools notices vulnerabilities in the application's run time environment for example by sending attacks. Examples: OWASP Zed Attack Proxy, Portswigger, Grabber, Vega
- Runtime application security protection (RASP) tools are used to detect attacks in application's firewalls and start the mitigation immediately. Example: OpenRASP
- Dependency scanning tools can be running in the build to notice if there are any known vulnerabilities in any of the third-party components. Examples: OWASP Dependency check, Node Security Project, RetireJS, OSSIndex, Bundler-audit, Hakiri, Snyk, Gemnasium, Source Clear

In addition to the automated tools mentioned above there are several security tools to test and monitor the communication networks and the infrastructure security, too.

4.3.9 Monitoring tools

Having both development and operations incorporating self-monitoring and analytics into the pipeline allows end-to-end monitoring of the infrastructure, pipelines and applications (Sharma 2017, 26). Monitoring tools offer various functionalities such as aggregation of all logs to a centralized system with a user-friendly interface and secure access controls, measurement of the latency of the servers based on the geographical location, and visualization, automatic alerts and proactive measures (Gonzalez 2017, 364–375.)
Examples of monitoring tools: Nagios, Raygun, Splunk, Pingdom Loentries, AppDynamics.

4.4 DevOps adoption

DevOps is not a clear process or methodology that can be taken as it is and implement. It is not a solution that fits to all organizations as it is, because the principles and practices

that come with it have an impact on people, processes and tools. It is a conceptual framework that provides principles and practices that can be embedded to an organization's processes to gain benefits, but it requires the organization to restructure the way it works. (Sharma 2017, 40; Erich et al. 2014.)

Wiedemann (2017) used staged model to analyze the extent of which DevOps is taken into use in organizations. Based on this approach, first a match between the innovation and the application environment must be identified before moving to the adoption of it. The organizations and the teams must first identify how they could benefit from DevOps before they are ready to move to the adoption stage. This identification requires knowing what is your starting point today and what do you want to achieve with DevOps (Sharma 2017, 41). Adoption is essentially about reaching the determination to apply DevOps within the organization. Adoption will be started with the training, development, installation and maintenance of DevOps -related practices and tools, and it follows with the ultimate decision to embrace DevOps as an everyday activity with commitment to continuous improvement.

DevOps practices (Continuous Integration, Continuous Delivery and Continuous Feedback) are comprised of several different technologies, skills, tools and methods, and the adoption of them puts an organization to go through a real transformation. The transformation must happen on all areas of organization. DevOps adoption can be observed at individual level, team level, or department level. At an individual level, DevOps can be seen as a specific personnel trait or a set of skills covering both development and operations. On a team level, DevOps adoption is about having cross-functional teams that are responsible for both development and operations. Finally, on a department level, DevOps adoption is about aligning the work of development and operations departments. (Sharma 2017, 104–105; Erich et al. 2017.)

Rolling adoption is a recommended way to start adopting DevOps. In rolling adoption, the organization starts with piloting DevOps in few projects, from which the learnings can be further used to adopt DevOps successfully all over the organization. However, adopting DevOps at scale requires more than just a bottom-up approach; in addition to bottom-up engagement it requires also a top-down approach with a well-defined strategy and objectives. The executive leadership must have the desire to change the culture and actively sponsor the DevOps transformation. (Sharma 2017, 64.)

Forsgren et al. (2018) emphasize that the DevOps adoption should not be measured following maturity models because they generalize teams' capabilities and technical proficiency and do not encourage continuous improvement. Also, the number of installed DevOps tools does not yet tell about how maturely the organization takes advantage of them. Therefore, the researchers suggest focusing improving the proven capabilities that lead to higher performance.

In the State of DevOps report from 2018, the researchers distinguished the stages along the organizations' DevOps adoption. (Figure 8) In the foundational stage (Stage 0), the organizations start building the foundational practices that are further adopted and enhanced during their journeys towards DevOps. Building these foundational practices includes implementation of relevant tools and processes that enable DevOps. When the basis for DevOps adoption has been built, the organizations can expect to see the first stage (Stage 1) of DevOps evolution. In the first stage the teams start to normalize their technology stack so that the agile development methods and tools such as version control are in daily use.

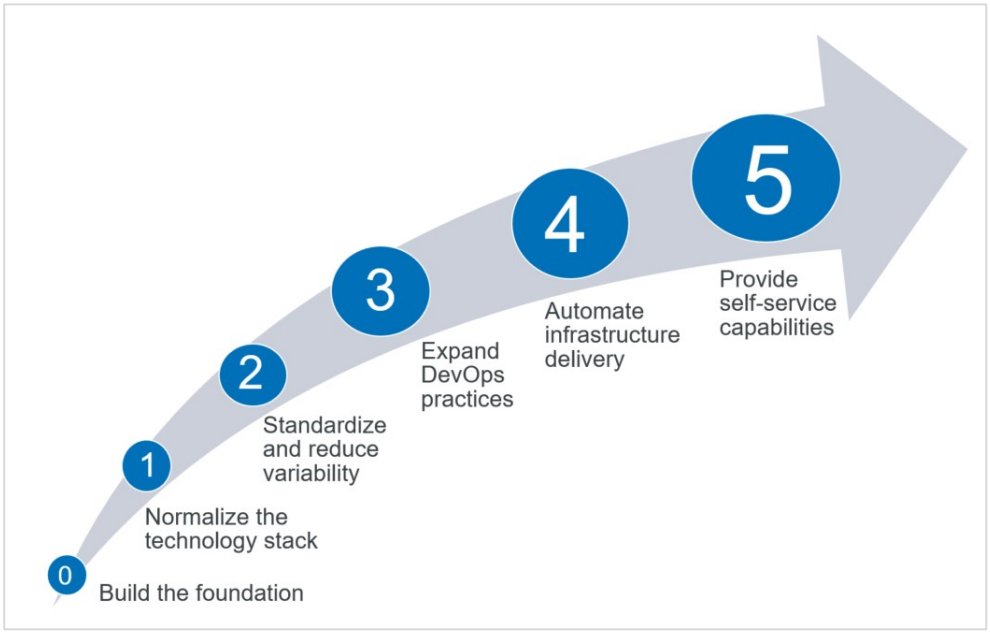

Figure 8 The stages in DevOps adoption (adopted from State of DevOps report 2018)

Moving on to the second stage (Stage 2), the development and operations teams typically start standardizing the set of technologies in use, although often still without further cross-functional collaboration. The standardization of the DevOps toolsets help the teams to scale the existing skills to accelerate the speed of delivery, reduce errors and enable more consistent deployment practices. When organizations start to evolve towards DevOps and expand the practices at a fast speed, a cultural change and cross-functional collaboration is required to cope with the speed. In the third stage of the DevOps adoption (Stage 3), the organizations are faced with the reality where operations teams are unable to handle the speed of delivery and the IT organization is struggling with the time to market requirements or quality. That is where the cultural shift is required. To gain advantage from the efficiency and speed, the organizations must form the culture of trust between development and operations teams that gives freedom to the smaller teams to work without manual approvals from people who are not involved in the everyday work. The culture of

trust goes hand in hand with the adoption of testing practices that build quality in by ensuring the predictability and reliability of the changes.

In the fourth stage (Stage 4) the organization is evolved so that it can automate the infrastructure delivery. Having the system configuration and provisioning automated and in line with production, offers the possibility for operations to provide developers self-service and enable them to deploy faster and more securely. The State of DevOps report (2018) describes the final stage (Stage 5) in organizations' evolution to DevOps to include the ability to provide IT resources in a self-service manner including automation of the incident responses. In this final stage of the DevOps adoption the whole organization leverages the automation capabilities so that it achieves the efficiency and avoids any waste in the IT delivery, such as handoffs, waiting and delays.

4.5 Adoption challenges

Both technical (tools, infrastructure -related) and cultural (behavior, work style -related) challenges are reported on the DevOps adoption. Culturally, the organizations may face a challenge of overcoming the initial mentality of development versus operations. It is challenging to introduce new practices and tools in the teams as it often requires a change in the mindsets and attitudes. (Lwakatare et al. 2015.) Adopting DevOps in small teams or start-ups may require less of a cultural change effort. In large enterprises that have developed their culture over time there is a bigger challenge of overcoming the change resistance. In addition, the bigger enterprises typically emphasize governance structures and processes over the culture. (Sharma 2017)

One of the biggest challenges with DevOps is combining it with existing IT Service Management processes. Over years many organizations have established roles and Change Advisory Boards (CABs) based on the ITIL (Information Technology Infrastructure Library) methodology. ITIL is often seen as contradicting with DevOps approach that emphasizes agile development, cross-functional teams and fast feedback loops. In ITIL, manual approval processes are established for changes, and the changes must usually be documented and approved in some change management tool. The most radical difference in DevOps is that the change is managed continuously from the start of the development with the help of automation and tracking of the changes in the version control. (Gruver 2016, 14–15.) Also, DevOps practices that aim to remove all unnecessary change approvals and centralize the development and operations work may raise segregation of duty concerns into the picture (Bird 2016). DevOps and ITIL practitioners should work together to better understand how to integrate existing processes with DevOps; how the roles need to change and how the automated tools and practices can

support transition to more flexible change/release management processes. (Ravichandran et al. 2016, 134–135.)

For example, when developers are merging some critical changes to the configurations, the pull requests can be used by developers and release managers to review and approve certain changes to be deployed. (Verona 2016, 53–54.) This means that the release managers are involved in the development work along the whole software delivery cycle instead of only in the end stage. Sharma (2017) recognizes following three points that are critical to discuss in ITIL and DevOps alignment;

1. Making ITIL practices leaner by reducing the cycle times.
2. Reducing approval steps by incorporating policy and rule-based automation to the processes
3. Using telemetry from DevOps pipeline to comply with ITIL controls.

The change resistance may originate also from the developers, testers and operations practitioners as DevOps brings changes to their responsibilities and roles. Gall and Pigni (2018) carried out a case study on adopting DevOps and found out that the new ways of work were indeed not well received by the developers. In DevOps, the developer is not anymore only developing, but delivering the service. The testers are not anymore manually executing tests, but they are providing the test automation capabilities. And finally, operations teams are not anymore firefighting; building and provisioning servers, but they are providing the automated infrastructure. (Sharma 2017, 31.)

The constraints for DevOps adoption are not always in the hands of the people or the organization. The business environment may introduce challenges that must be addressed in DevOps adoption. For example, adopting DevOps practices in highly regulated industries is challenging when it comes to the documentation and reporting related requirements from regulatory authorities. As a response to the rigid software development practices in these industries, Laukkarinen et al. (2018) raise a need to develop DevOps practices that would be applicable across the standards in a regulated software development.

Typically, there are also suppliers involved in the IT delivery, especially in the enterprises where the IT has not traditionally been the core business. More often, the supplier for software delivery is different from the supplier for the operations after go-live. This supplier 1 vs supplier 2 type of problem introduces a challenge to the DevOps adoption as their incentives and goals specified in the contracts are usually different from each other (Sharma 2017, 303). Hering (2018) suggests that the chosen partner should be responsible for both software delivery and operations after the go-live to ensure smooth continuity. The joint culture with the partner should be formed together by using agile contracts.

Technical challenges may occur for example when too much focus is put on the tooling. The integration and maintenance of all the tools, and the move from legacy infrastructure towards infrastructure as a code may become more difficult than expected.

(Ghantous & Gill 2017, 6.). Based on the study by Gall and Pigni (2018), developers found that there is a need for specialized knowledge related to the maintenance of the CI/CD pipeline, which the developers rarely had.

Adopting DevOps drives organizations to implement automated tools to replace the manual error-prone work. However, the automation does not come without challenges. Usually, a human resource and time must be invested for the initial, manual programming of the automation scripts. This also introduces a need for extra checks for quality and security, as there is a risk that the automated code may sabotage the whole application, the tools and the configuration. (Sharma 2017, 293.) Moreover, also the automation capabilities must be maintained and developed.

One of the biggest technical challenge that organizations may face when adopting DevOps is the architecture. The IT architectures usually reflect the organizational needs and as the organizational needs change, the companies need to go through some architectural transformations as well. Many big organizations have complex, tightly coupled architectures in which every small change has a dependency. These monolithic architectures pose a challenge to DevOps adoption, as DevOps is optimized for loosely coupled service-oriented architectures or microarchitectures that allow scaling, reusability and automated, frequent and safe deployments. (Sharma 2017, 248 ; Balalaie et al. 2016 ; Ebert et al. 2016 ; Kim et al. 2010, 181–189.) The decoupling of the architecture is necessary because today's architectures should be designed for elasticity; continuous evolution and change (Hering 2018).

5 CASE STUDY: DEVOPS IN IDENTITY AND ACCESS MANAGEMENT

5.1 Case study design process

This case study was conducted in a multinational corporation (Company X). The application domain for the case study was the Identity and Access Management team of the company. The case study lasted for five months in total and it followed a design process approach suggested by Zimmerman et al. (2004) (Figure 9). By following this approach, the case study had six iterations.

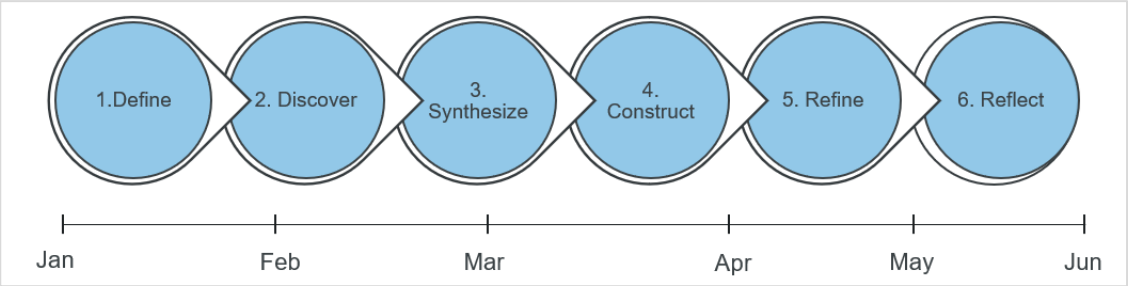

Figure 9 Case study Design Process

The first iteration of the design process was Define, and it included the definition and assessment of the case company. This was done during January until beginning of February. Next iteration was Discovery, and it refers to the field work during which the context and the requirements are explored. This was be done by conducting analysis on the topic in the environment. The Discovery was started already in January and it was finished by the beginning of February. The third iteration Synthesize was started in the beginning of February, and it included the researcher exploring the points of friction in the application domain. This iteration lasted until mid-March. The next iterations Construct and Refine included the actual design work, during which the researcher generated and evaluated the design iteratively building on the knowledge base and the meta-artifacts found during Synthesis. This design phase was done between mid-March until May. The last iteration of the design process was Reflect, which lasted from beginning of May until the end of May. Researcher reflected on the case study and evaluated how the design was received in the application domain.

5.2 Data collection process

The data was collected during five consequent months in the form of literature review, interviews and document review/observation in the application domain. The data collection process was continuous, which is natural to a design process. The design process was all the time influenced by the environment of the application domain and the knowledge base. Literature review was conducted throughout the process, but more intensively during the first three iterations of the design process: Define, Discover and Synthesize. The document review and observation were used as data collection methods from the application domain constantly during the whole study, with a deeper focus during Synthesize and Construct iterations of the design process. It was found particularly good technique to actively observe and review data from the application domain to gain understanding on the existing processes and thus help the construction of the proposed design. The interviews were held in the beginning and end of the study in iterations Define and Discover to better understand the context of the study, and in the end during the iterations Refine and Reflect to collect feedback for the design evaluation. The Figure 10 illustrates the data collection process in line with the stages of the design process between January and June.

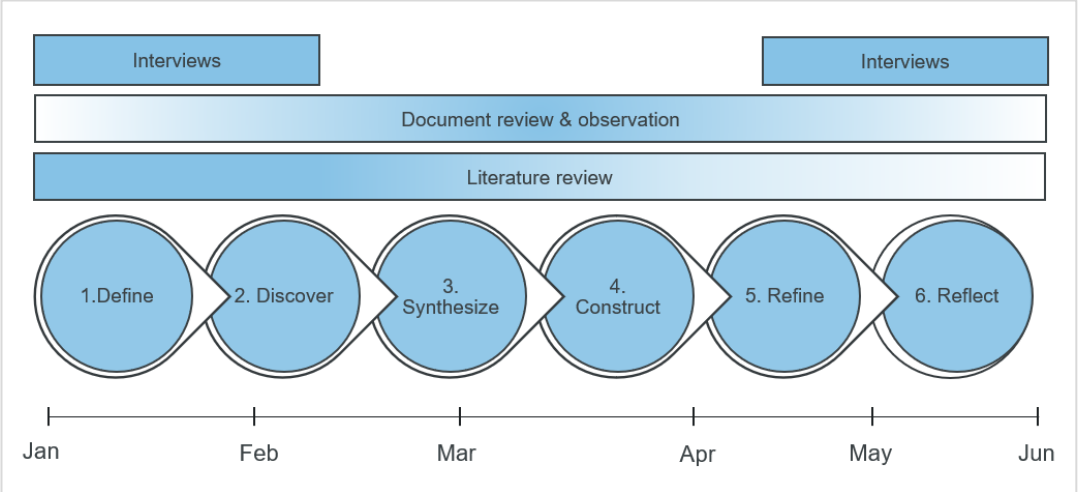

Figure 10 The data collection process mapped with the case study Design Process

5.3 Introduction to the case company

The Company X is a publicly traded corporation specialized into developing, engineering and manufacturing industrial products and services with over 50 000+ employees in more than 60 countries globally. The technological disruption in the areas of computing, mo-

bility and connectivity combined with the fast speed of change are impacting the company's business, the lives of its customers and the ways of working. One of the strategic risks of the Company X today is the inability to quickly respond to the customer needs and not being able to deliver new solutions and services to the business as fast as would be required for success. The changes in the business environment have led the company to initiate transformation programs in all areas of the organization. Few years back, the company launched a strategic program to transform the company into an agile organization by putting a customer in the center of all development. In recent years, Company X has also started to embrace new technologies to innovate new digital services and products. One of the significant efforts has been to combine **R&D** and **IT** functions into one global **Technology unit** to improve the company's development in the era of digitalization. Inside the Technology unit, Company X has a typical bimodal IT function.

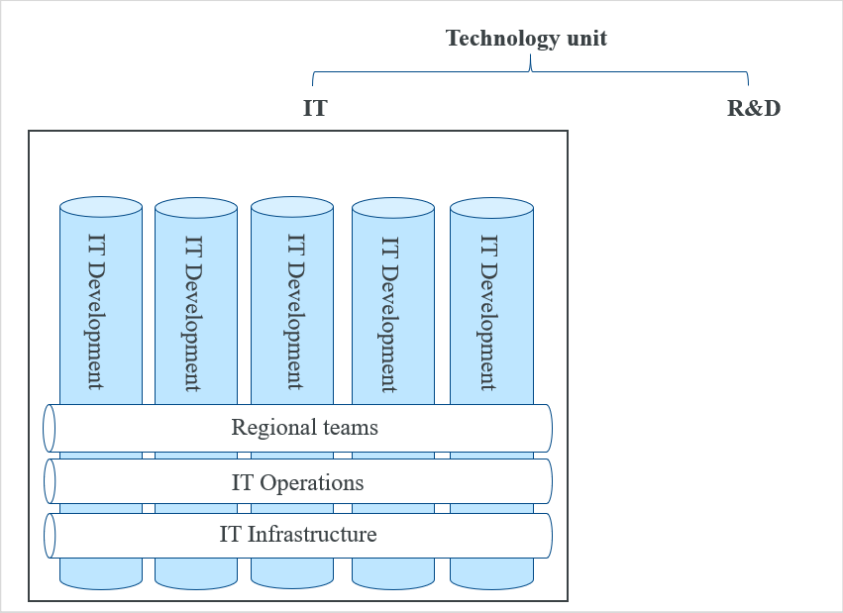

Figure 11 The organizational structure of Company X's IT unit

The explorative, experimenting IT that innovates for new customer solutions belongs to R&D and the IT solutions that keep the business running belong to the IT. The role of IT unit is to support the company's business globally and provide IT expertise to R&D. However, the IT unit also seeks to constantly improve and introduce new innovative and efficient 'keep the business running' solutions and deliver value to the business. The strategic goals of the IT unit are agility, stability, cost-efficiency, right time-to-market and better user experience. These goals are driven by customer centric thinking, collaboration and innovation, new competencies and fast, smart execution.

The IT unit has approximately 400 employees that are located in more than 20 different countries. The unit is organized into five vertical **IT Development units** that have multiple teams. These teams design their IT solutions based on business requirements, and have the ownership over the solution's demand, development and service processes. The two horizontal teams are **IT Infrastructure unit** and **IT Operations unit** whose role is to support these solutions teams. IT Operations unit's role is to offer maintenance and support services to the development, testing and release of the solutions. IT Infrastructure unit is responsible for ensuring the quality, stability and usability of the IT infrastructure, global services and support. On top of all, there are five **regional IT teams** that work to ensure local IT service delivery.

5.4 Introduction to the application domain

Agility that the companies seek for today requires more effective and efficient management of digital identities and their accesses to resources (Bertino & Takahashi 2010, 13). Identity and Access Management team of Company X belongs to IT unit. The purpose of the IAM team is to provide services for digital identity management, authorization and authentication with a goal to improve Identity and Access Management aspects of cybersecurity in the company. The first downstream customers of IAM are all the applications that are used in the company. The IT solutions provided by IAM team are the identity management system (IDM), group and account management in Active Directory (AD), Single Sign On (SSO), password reset and multi-factor authentication (MFA).

5.4.1 The team

The team consist of four internal team members. IAM Solution Manager leads the team and has the end-to-end responsibility of the solution including requirement management, planning roadmaps, development, roll-outs, support, maintenance, continuous improvement and retirement. IAM Solution Manager is also responsible of the solutions' quality and costs. In addition to the IAM Solution Manager, the team consist of two Design Managers and one Analyst. The Design Managers are responsible for the key user network, managing requirements and turning them into a solution design. In addition, Design Managers play a key role in supporting IAM Solution Manager with the solution road mapping. The analyst's role is capturing requirements and creating concepts into designs.

5.4.2 The problem statement

The business drivers defined for IAM are security risk management, operational excellence and user experience, of which security risk management is the foundational key business driver. IAM solutions aim to strengthen Company X's security risk management by providing 360-degree view over users' accesses and activity, regular access reviews and removal of unneeded accounts, accesses and rights, and enforcement of segregation of duty. IAM also aims for operational excellence by streamlining and automating manual processes, reducing amount of service desk calls/tickets and consolidating IAM infrastructure and software. Finally, the solutions enable the business by delivering smoother and faster user experience.

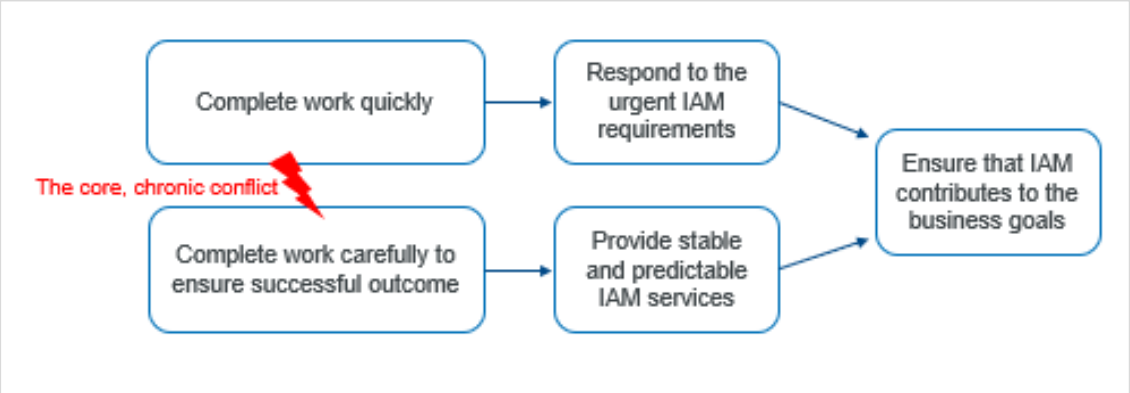

Figure 12 The core, chronic conflict in Identity and Access Management

Business value from Identity and Access Management is measured by how many applications in Company X have appropriate identity management, authorization and authentication in place. In other words, how many applications are IAM compliant. To have appropriate identity management and authorizations in place, all applications must be onboarded to the IDM -system and have SSO and MFA enabled. One of the targets of IAM in 2019 is to make all these IAM deployments self-enabled. IAM team is driven by the demands from the business and thus needs to fill these targets at a fast speed without influencing the availability or quality of the daily operations. The Figure 12 demonstrates this core, chronic conflict that is present in the IAM area.

Company X's IAM team is at the moment of the case study developing and implementing a new IDM solution and wants to adopt DevOps principles and practices to their work to generate business value from it fast, while ensuring high quality. DevOps practices have been taken into use by some other teams in the organization, however effective knowledge sharing has not been in place yet. The IAM team does not have expertise on what is DevOps and how the principles, practices and tools could be adopted in the Identity and Access Management area.

6 ANALYSIS AND DESIGN CONSTRUCTION

This chapter will start with an analysis of what is the level of DevOps knowledge and what are the challenges in the application environment (Chapter 6.1). After, the friction points in Identity and Access Management are presented with designed propositions of how to adopt DevOps to fix those gaps (Chapter 6.2).

6.1 DevOps in the IT unit

DevOps in the IT unit –analysis was made based on conversational interviews that were conducted by the researcher in the organization. The scope of the analysis was restricted to the IT Development teams and IT Operations teams. These units were chosen due to their position to developing, testing, deploying and running services. The purpose of the DevOps discussions was to collect data from the application domain with main goals being to a) discover what is the general knowledge level around the topic in the context, and b) gather further knowledge and insights on the topic. Therefore, sometimes a participant was chosen from a team that has more experience from DevOps.

The following pattern of questions was flexibly used as a question base to conduct the semi-structured interviews. The chosen approach gave flexibility for both the participant and the interviewer to skip questions if needed, go through the questions in different order or ask follow-up questions where possible.

1. What is your background to DevOps?
2. If you have experience on DevOps
    a. In what kind of projects?
    b. What tools where in use?
3. What do you see as the challenges when adopting DevOps?
4. What do you see as the success factors when adopting DevOps?

The interview participants were contacted mainly via email with a proposition to discuss the topic. The discussions varied from 30 minutes to over 60 minutes in length based on the participant's knowledge level, experience and personal interest. In addition to English, Finnish was used to conduct interviews, and those responds were translated to English. Altogether **25** persons provided inputs to the analysis, from which 2 gave their inputs via email. Around **68%** of the participants were from IT Development teams, including at least one participant from each of the five units. The remaining **32%** of the participants were working in the IT Operations unit. Most of the participants were IT Solution Managers, Directors, or Test/Release/Service managers.

6.1.1 Knowledge level

The participants were categorized into following different groups based on their level of knowledge on DevOps.

- **Group 0 – No understanding** includes participants that had never heard of DevOps or had heard of it but were not familiar with the concepts or practices of DevOps.
- **Group 1 – Low understanding** includes participants that had heard of DevOps before and were somewhat familiar with some of the DevOps concepts. The participants that were grouped to this group had understanding mostly on the agile but had difficulty to understand the difference between agile and DevOps.
- **Group 2 – Moderate understanding** includes participants that were familiar with the key concepts of DevOps, had been learning more by themselves about the topic, and mentioned that they had seen some DevOps practices in use during their career.
- **Group 3 – Good understanding** includes participants that had good understanding of the key concepts of DevOps, combined with extensive self-learning or training. Some of the participants in this group also possessed some experience or skills from one or more areas of DevOps.
- **Group 4 – Excellent understanding** had very good understanding over DevOps concepts and tools combined with extensive self-learning, trainings and experience of DevOps adoption during their career.

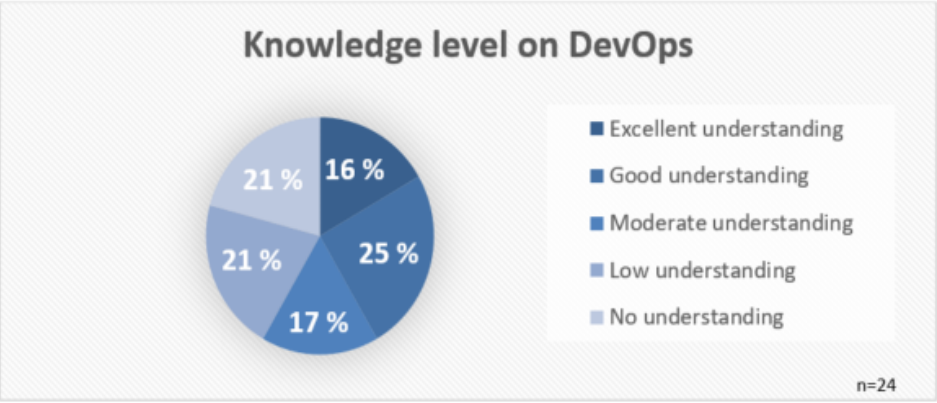

Figure 13 Knowledge level on DevOps (n=24)

The Figure 13 illustrates the division of knowledge level among the coverage of 24 participants (Note: one participant provided only general thoughts that could not be used to assess the knowledge level). Based on this categorization, **42%** of the participants had low understanding or lacked any understanding on the topic. These participants stated

that they did not know or have a clear idea on what DevOps is. Half of them had heard 'agile' and 'DevOps' terms used inside the company, understood the main principles of agile but did not understand the difference to DevOps. Most of the participants in the groups 0 and 1 had a strong background working mostly with inflexible ERP systems with many dependencies, and therefore, they had difficulties to understand how agile practices could be applied at all in their own area.

One participant that had background in legacy IT recognized that "*We do not have any knowledge even on agile, not to mention DevOps."* Some referred to these terms as 'buzzwords' and felt afraid to rush into adopting new practices without understanding what they really mean and what do we want to achieve with them. One participant stated that *"the biggest challenge is to avoid chaos",* and another mentioned that "*we need to first understand what we are even doing before adopting DevOps"* Also, the participants seemed to strictly apply ITIL methodology in their daily work and had difficulties to step out of the framework and question their current ways of working. One of the participants recognized this by stating "*We need to become aware of what is DevOps and how to get there, but we are so much biased on what we know that we are not open to new. We should invest time and commitment to think about how we can do things better."*

Participants in Group 2 **(17%)** had moderate understanding on what DevOps is, and they could describe the basic ideas around DevOps. All the participants that had moderate knowledge level had been self-learning about DevOps. Also, all had some previous experience outside of Company X from at least some DevOps practices such as CI, test automation, management of teams with Dev and Ops personnel, or software development experience in a DevOps team. One of the participants also reflected 20 years back in history and recognized that the work was more "DevOps-a-like" back in then: *"I figured out 20 years ago that what we do is taking requirements and develop services to operate them locally without centralized governance, in a team of 10 people. We did not have the tools of today, but we had people involved that both developed and operated."*

In this category the participants had been thinking more of the topic and had started slowly re-thinking the way they work. One commented "*What is the role of release management anymore if everything can be continuous? It feels like we can work in agile way, but then we need to spend time on this theater of change advisory boards on the side."* Another participant commented also that their work is restricted by manual test and release schedules that do not consider automatic deployments.

More discussions were held with participants that had good (Group 3) or excellent (Group 4) understanding on DevOps. The main difference between these groups was that the participants with excellent understanding on DevOps had some experience on adopting DevOps. However, the participants in Group 3 **(25%)** had been attending DevOps trainings or read extensively on the topic compared to the participants with moderate knowledge. These people also mentioned that they are planning to adopt DevOps in one

way or another in their own area of work. The participants with excellent knowledge in DevOps represented **16%** of the all participants. These individuals had profound understanding on the history of DevOps, the principles and the key practices. These people also had experience from DevOps adoption; some had experience more specifically from the tool -side of DevOps. These participants could describe the tools needed for building the DevOps pipeline and could specify challenges, success factors and specific thoughts on

DevOps adoption in much more detail than the participants from other groups. Important to notice is that most of the participants with good or excellent knowledge level on DevOps were working for specific teams in Company X; either in IT development of customer facing IT solutions, or in IT development for R&D. Also, the discussions revealed that in R&D, the level of test automation is high and approximately 40 teams have some DevOps tools already in use. However, the level of actual use of the tools is not known. All in all, these initial findings strengthen the assumption that there has not been effective knowledge sharing across organizational boundaries. The results indicate that almost **60%** of the participants from IT unit did not yet have a good knowledge level on DevOps regardless of the seemingly high adoption rate of DevOps in R&D, or the DevOps experiments elsewhere in the organization.

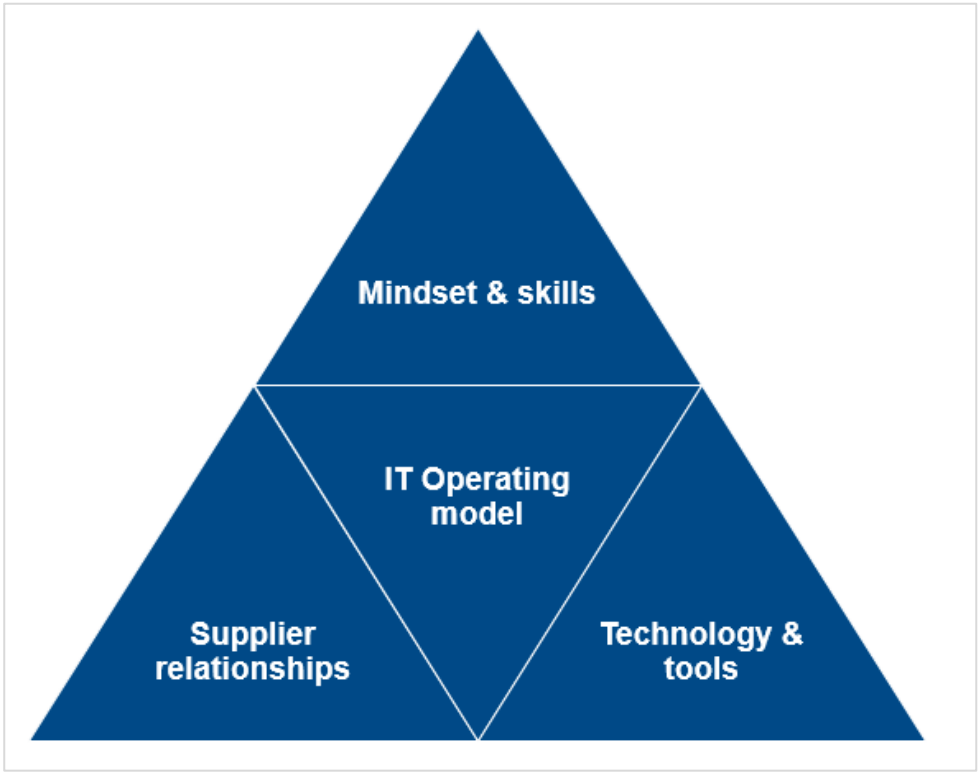

Figure 14 The four main challenges with DevOps adoption

When it comes to the DevOps enablers, one participant noted that "The challenge is to scale DevOps tooling beyond the R&D because R&D infrastructure is separated from IT

unit's cloud infrastructure." The security issues were raised as the main inhibitor for reutilizing R&D's DevOps resources in IT unit; R&D operates almost without exception in the internal network. The four main themes that were raised as challenges among participants when discussing DevOps adoption were the IT Operating model, supplier relationships, technology and tools, and mindset and skills (Figure 14). In the next sections, these four main problem areas are discussed more in detail.

6.1.2 Challenges

1. IT operating model

The current IT Operating model and existing silos were highlighted as the main challenges for DevOps by **40%** of the participants. It was mentioned that the IT operating model is based on the waterfall model and has too strong focus on processes, roles and specialization. It was also mentioned that the model neglects the fact that all the work in IT is somehow dependent on each other, and that some of the processes could be automated with new technology. One participant from IT Development described that the current way of working creates waste to the processes and that there is a barrier between IT Development and IT Operations; *"We develop something in our team and when we are ready we just go to the support team and tell them that 'we came up with this, you should operate it'"*. Another participant noted that everyone seems like focusing only on their own area without thinking about the whole value stream.

One of the IT Development units has kept the testing and service management resources on their area instead of having resources coordinated by IT Operations, whose role is to support the development, testing and running the service. This is because the IT Operations unit has background as a unit that is building standardized common enablers, outsourcing the competencies and targeting for economies of scale. One of the participants stated that "*The idea behind separate Operations unit is very "old school", and based on the factorized IT. It creates barriers between internal teams and our partners in development."* Also, another participant from Operations unit recognizes the challenge of combining the benefits of scale with DevOps: *"In order to adopt DevOps, Operations unit needs to ensure the Ops resources in Dev teams. Currently, we cannot have the dedicated person from service operations to sit with the development team due to resourcing decisions."*

In addition to the silos between IT Development and Operations teams, some barriers for DevOps adoption are mentioned to exist between enterprise architecture and rest of the IT, and portfolio/project management and IT. Enterprise architecture's role was emphasized to be important when making tool-related decisions while adopting DevOps. It

was mentioned that *"We have lack of domain experts that can see the whole picture for building the solutions"* and *"The enterprise architecture is now working in a silo. The domain architects should work more together with the development and operations."* However, the participants acknowledged that the involvement of enterprise architects is a matter of resourcing.

It was also mentioned that the current portfolio/project management process does not support DevOps adoption, as it is not encouraging experimenting and failing fast: *"Our portfolio/project management is not agile, and it is difficult to get financing for an innovation."* Also, it was mentioned by another participant that *"Business has a mentality of "what can you deliver and when and with how much money?" The business makes money decisions project based without considering the operational costs that occur after the project."* This mentality combined with the Development and Operations working in separate teams is seen to create a wall of confusion. It was mentioned that the business requires something new to be developed and when Development teams finish their development projects, they just hand the developed service over to Operations teams that carry the cost of running the solutions.

2. Supplier relationship

Company X has outsourced its development, testing and support of the IT solutions to the extent that in most of the solutions only the design work is done in-house. This decision has been made mostly for centralization and cost reductions reasons. The supplier relationship and contract management became very much discussed topic in terms of DevOps adoption. **28%** of the participants had serious concerns on the topic.

It was mentioned that IT unit is very dependent on its suppliers and therefore, in DevOps adoption it would be important to re-think what Company X expects from them, how Company X wants the work to be delivered and how to enforce the DevOps culture with the suppliers. One commented *"I believe the biggest challenge in corporations that want to adopt Agile and DevOps is that when they outsource development, the contracts do not define the way of work; developers are just paid to develop. The incentives for the work should be thought again when adopting DevOps".* Some participants specifically mentioned that even though Company X had decided across the organization to adopt DevOps, the DevOps ways of working cannot be expected from suppliers unless the suppliers are actively engaged into the new model and trained about the practices, tools and cultural change.

One person presented also an opportunity regarding supplier selection; *"The fact that we want to move to DevOps mode of working can be also boosted by our suppliers, especially the start-up minded companies or companies with niche expertise. They work pro-*

actively in DevOps mode; they are autonomous, collaborative and agile." Some participants that had previous experience in DevOps adoption noted that also the cultural and geographical factors affected the DevOps adoption as building trust and common culture with suppliers took time. One mentioned that it took the team 1,5 years to eventually reach the DevOps mode of working with their supplier.

3. Technology and tools

When it comes to the technology and tooling, there were complaints about the lack of visibility over the tools that are in use across the company Three Solution Managers mentioned that they have no visibility at all to any tools used in development, testing or deployment of their solution because everything was decided and done at the suppliers' side. Some complained that they would be interested to adopt DevOps tools but had no visibility over what tools other teams have used or what could be used and how.

Many thought that some standardization over the DevOps toolset would be required. One participant mentioned that *"It doesn't make sense that everyone has different deployment tools, different monitoring tools... it just ends up in a spaghetti of tools. The common way can be DevOps but there needs to be common tooling across the company.*" Another participant also pointed out the need for common tools and practices, but still recommended to leave space for solution specific tool decisions*; "It would be great to have company-wide DevOps best practices and recommended tools to support DevOps adoption. However, the tools should not be pushed on the teams as not every tool fit into every solution. Rather, offer a recommended toolkit from which each team can choose upon."*

Other tool related improvement points that were brought up repeatedly in the discussions was the need for automated configuration management and move towards infrastructure as a code. Many teams did not have version control in place and complaints were raised about missing a standardized proper requirement management tool and a centralized place for all design documents. Additionally, the low level of test automation in IT was seen as a roadblock for continuous deployment, **24%** of the participants emphasized that the level of testing is still in early phases. One Solution Manager stated that they are missing a clear testing strategy and the testing is not organized well enough.

However, some participants had already been directly involved in developing test automation in the company, and one of them mentioned that *"By adding test automation we can avoid errors. Automation gives us confidence on our software quality and our experts can focus on more value-adding work. We want to be convinced that CI works and gain experience with test automation before moving to continuous deployment mode. However, the down side is that we also need time to maintain the automation."* It was wished by the participants that the upper level management of Company X would consider infrastructural changes and investing time and resources to automation efforts.

4. Mindset and skills

Some of the concepts that were repeated by many participants when discussing the other-than-tooling related success factors for DevOps were collaboration, autonomous teams and common targets. In general, the participants think DevOps is not just about plugging in tools and using them, but instead it is a change management effort that should be acknowledged. One participant noted that *"You can't tell IT to "be DevOps", it is about changing the way of working, it is change management."* It was also mentioned that for adopting DevOps people must change their mindset towards becoming open to new ideas, learning and being ready to work in the same team with people from variety of functions; *"The DevOps skillset should not mean that you have to know how to do everything but you need the ability to discuss and understand effects of others' work."* It was mentioned that DevOps requires re-thinking the impacts that your own work has on the whole value stream. One person recognizes that it is challenging to have people change the ways they are used to work while being constantly pressured by the speed; *"In Company X we have a culture that we do everything at the same time. Even though the new way is introduced, many people turn into old ways of doing under pressure. Management needs to understand that going through change will take time and slow us down for a while."*

6.1.3 DevOps Centre of Excellence

Based on discussions, years back in Company X's R&D unit when the first agile projects were started there, the teams begun to install DevOps tools with a help of external consultants. However, soon they realized that they had made a mistake of going "tools first" before considering the other organizational/process changes that might be required. All the teams' time was eventually spent on the maintenance of the pipelines and tools. As a result, a "DevOps services" team was unofficially formed to centralize the expertise and help the R&D teams to quickly setup the tools and provide help and maintenance with them. This team was (and is, at the time of writing) composed of external consultants that had worked for years with Company X in a deep partnership.

All the participants of the DevOps discussions could come up with long lists of challenges regarding DevOps adoption, however, also improvement ideas similar to the one above were raised along the interviews. "DevOps Centre of Excellence" -type of idea was raised by five different participants. The participants stated that the teams should not have to start DevOps adoption from the scratch, but the existing expertise should be centralized inside Company X to scale the DevOps practices, tooling and the maintenance of the

tools. Also, the DevOps Center of Excellence could standardize the tool chain, provide training about DevOps culture and the related best practices.

Some suggested that the IT Operations unit should change their ways of working from the old school IT towards becoming DevOps enabler and a resource pool for DevOps teams. One participant commented: "*Standardizing of the inputs and outputs is required for DevOps to work smoothly. I see that IT Operations unit could enable economies of scale by providing DevOps resources to the projects."* Finally, as some people mentioned, the agile, DevOps and similar buzzwords often make people think about chaos. But, with a clear strategy and vision on how to get there we may realize that *"agility is well-coordinated discipline, not chaos"*, as mentioned by one of the participants.

6.2 DevOps in Identity & Access Management

6.2.1 Waterfall -based IDM service delivery model

Company X's IAM team is developing and implementing a new IDM system, SailPoint IdentityIIQ (IIQ), to provide centralized, automated lifecycle management and access provisioning of identities in the company. IIQ is a software package that can be configured and customized by IAM team to meet the business needs. The team had gone through one failed IDM system implementation previous year. The implementation failed due to a technology misfit which became unbearable in the testing phase of the project. The project followed the enterprise-widely used standardized gate-based waterfall development model. The model is based on the traditional waterfall method comprised of four different phases, nine stages and six quality gates. In order to pass through these gates, the project team needs to wait for approvals and sign offs from IT management team, project steering and portfolio/investment steering. In addition, dozens of different deliverables need to be prepared and handed in to pass the gates before the IDM can finally be deployed to deliver value to the business. This development model introduces waste and builds up silos between different teams and activities.

The earlier IDM implementation project started by designing the system based on given business requirements. The functional design work was done by the internal Development team and rest of the configuration, customization, testing and future operations work was outsourced. IIQ development was provided by Supplier 1, and testing and support by another Supplier 2, and both teams worked from geographically different locations. The whole implementation project was supervised by an external project manager, however the internal IAM team had an active role in the implementation related coordination and preparation.

This mode of working introduced four friction points to the work. The first friction point can be found between the internal solution team and external IDM development team. The IAM team expected that the supplier's development team implements the system based on the design, while the supplier had difficulties to build the system without having profound expertise of the solution design and the underlying tool limitations. The waterfall mode of work did not allow the team to continuously design, build and test the requirements in agile manner.

The second friction point is related to the wall of confusion between the development and testing teams. These activities were managed by two separate suppliers with different incentives. The developers focused on configuring and customizing the IDM system according to the design, without incentives to build quality continuously into the product. The testers were solely testing the software having no developing skills, and without being familiar with the environment setups or the code itself. The communication and collaboration between the teams was bad because the development was not prepared to test what they had created, and the testers did not proactively work to fix the errors and bugs they found during the testing. In addition, all the development and testing was done manually, which introduced slowness, unreliability and un-trackable errors to the work.

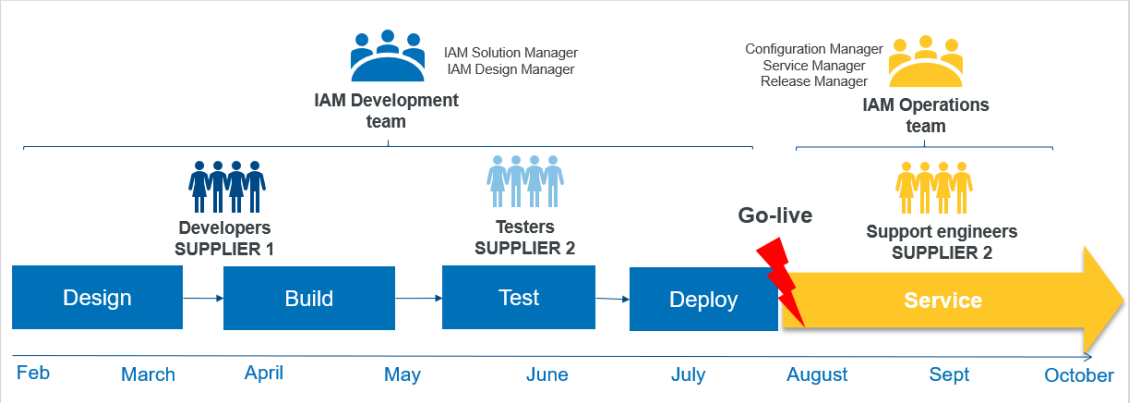

Figure 15 IDM delivery in the waterfall model

The third point of friction appears between IAM Development team (in blue in Figure 15) and the IAM Operations team (in yellow in Figure 15). After the testing phase the system is prepared for production deployment. Typically, it is only at this point, when the production environment will be ordered and installed. Also, the Maintenance and Operations Guide is supposed to be prepared by the people in the development team. The development team is not necessarily experienced in the operations side of IDM and has no responsibility over the service continuity after the go-live. In operations, the separate team would need to take care of the support and maintenance of the IDM solution following this Maintenance and Operations Guide that may not provide enough information to solve

all issues that may occur. Further, the IDM Service Manager and Release Manager are situated in IT Operations unit and are not actively involved in the IDM development activities before the transition of the development project to the service operations.

Finally, the fourth friction point concerns the handover of the IDM project from the Development team to Operations team, which is responsible for running the service. In this phase, the whole readily developed, configured and customized system is deployed all at once to the operations. When the IDM system is 'production ready', the IAM Development team needs to make sure all deliverables are ready and fill in the operational readiness form. After, the team must wait for approval from IT Management, project steering and portfolio/investment steering before the IDM can be deployed to production. These external approval processes present a classic friction point into the work. All the decision-making parties are very distant from the daily development work and are not familiar with the IDM system itself. Therefore, these decision makers will have difficulties to address all the risks involved and assess the quality of the product.

This research argues that Company X could have saved time and money during the project if the team had been following DevOps principles and practices that encourage to fail fast and deliver the IDM quicker in smaller iterations with the help of automation. An effective design can be built to guide the adoption of DevOps principles, practices and tools in the new IDM implementation project and the future continuous service delivery.

6.2.2 Continuous IDM service delivery model

After the friction points have been identified, the next step is to re-design the IDM delivery model in a way that friction points are removed, and the mode of working supports cross-functional collaboration, feedback, and continuous delivery of the IDM service. Part of this step is to adopt agile ways of work in which the work is broken into smaller items, done in smaller iterations, and new techniques are brought to the daily work to enhance visibility and traceability of the work.

Firstly, the teams should be reorganized into the IDM Product Owner team and the IDM DevOps team so that any friction between the parties involved can be minimized. In Company X, the resources for development, testing and technical support in operations are typically outsourced from one or several external partners, while the internal Development teams focus on Company X's business requirements and translating the requirements into IT solutions. However, in DevOps model, the development, testing and running operations will be ideally procured from the same supplier to minimize any contractual or organizational friction among the team members. The Product Owner (PO) team should include all people who are responsible for the development and operations of the

IDM service in Company X side, regardless of where they sit according to the organizational structure. The PO team does continuous planning by managing IDM related design requirements, prioritizing them, and breaking them down into simple, easily understood user stories with well-defined acceptance criteria. These work items are then assigned to the backlog that the DevOps team will work upon. Ultimately, the PO team's job is to think about "What are we building?" and "Why are we building it?", while the DevOps team focuses on "How are we building it with a built-in quality?".

The DevOps team is a cross-functional team that possesses competencies in development, testing and IDM operations and support. This team is coordinated by a person who will facilitate the daily work and coach other team members (e.g. Scrum Master). The DevOps team is given freedom to develop, test and deploy the backlog items autonomously with a common goal ("How are we building it with a built-in quality?"). Building the new continuous service model can be started by enhancing structure and visibility to the work with the help of agile methodologies such as Scrum. First, the work is scheduled into cycles that are of one or two week(s) in length, during which certain work items will be built, tested and deployed. A new cycle keeps starting with a PO team prioritizing the backlog. Then, the DevOps team gives estimation on the required effort for each item and informs PO team how many of the items they will commit to complete during the cycle.

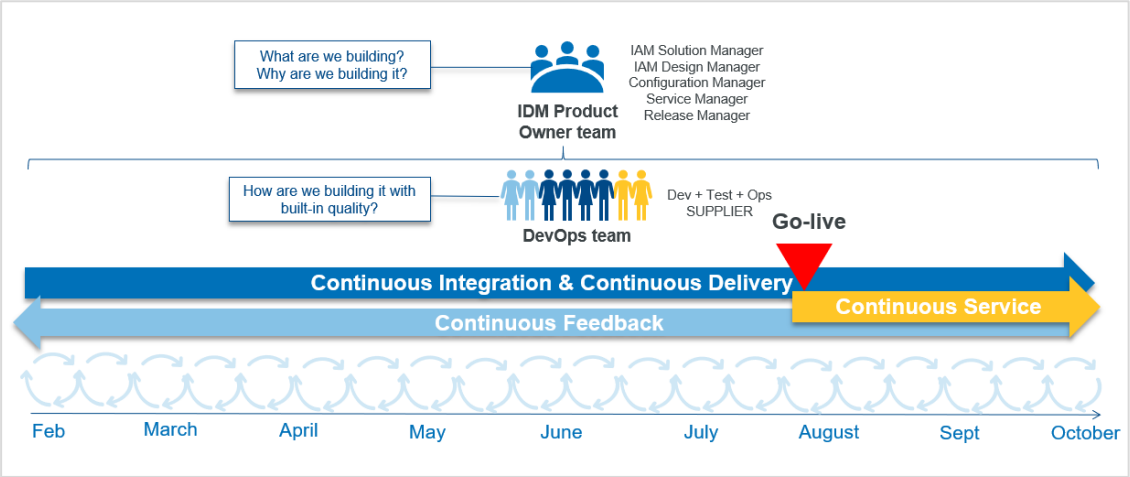

Figure 16 Artifact 1: IDM delivery as a continuous service

The status of everyone's work is gone through briefly in the daily stand-up meetings of 10 to 15 minutes in length, where any impediments or dependencies may be discussed, and the capacity may be adjusted if needed. Using a shared task board, the DevOps team members may easily assign tasks to themselves and others. The task board should also have a central role in the daily meetings by providing visibility over the parallel work items and offering a possibility to discuss and link the items. Each user story and task must have an owner who has the overall responsibility on the completion of the work.

The status of the work items is updated by the owner throughout the cycle for example "New → Work In Progress → Done". Finally, all the completed work is demonstrated to the PO team at the end of each cycle. After the work is reviewed everyone will sit down for a while to discuss the past cycle giving remarks on what was gone well and what did not go well. No one should be interrupted, and no one should judge other people's opinions, because every point of view is considered and accepted. Everyone in the team should suggest action points to be taken in the following cycles to improve for example the communication, the methods and/or the tools.

In each cycle, the team will develop and deploy IDM in small increments that can bring already some value to the customer. By all the time testing and deploying work to production, or pre-production environment, the team can ensure that the final go-live of a new service will be as smooth as possible. To enforce DevOps principles also in the development work, the IAM team needs a proper deployment pipeline for IDM. The deployment pipeline can be built by establishing Continuous Integration and Continuous Delivery with integrated Continuous Feedback mechanisms.

6.2.3 IDM deployment pipeline (CI/CD)

During this case study, the web-based Java application **SailPoint IIQ** will be built, tested and deployed by using a CI/CD platform called **Azure DevOps.** Azure DevOps enables the team to automate the build, test and deployment of the product. The deployment pipeline will solve the friction caused by manual development, testing and deployment. In addition, it will make the team's work more efficient when the most cumbersome and error-prone work is automated and made repeatable, fast and reliable. The integrated pipeline will also add collaboration and communication between the team members and increase visibility of the work flow from development all the way to operations, wherein the IDM will run as a continuous service.

To enable Continuous Integration, each team member needs to be trained and feel comfortable with using version control to store every artifact from configuration files to the customization code and test scripts. Changes are not allowed to be made from the user interface because they would be difficult to track and resulting bugs would be challenging to trace afterwards for debugging.

Azure DevOps can be connected with a **Git** repository where all the IDM related configuration and customization code can be stored and versioned. The configuration and customization files will be stored in their respective locations in the version control together with all environment specific build configuration files. Using Git, the IDM DevOps team can clone the repository to their local machine and start working on the

configurations and customizations on their own development branch with a chosen development tool.

For each piece of code, the developer needs to write unit tests to test the behavior of the change they make. All the unit tests should be stored in a shared source folder in the repo, and the test rules should be stored in the specified rule library. After the developer has committed the changes and unit tests, she/he is responsible to push the changes from her/his own branch to the master branch and always delete the development branch after. Development branches should be as short-lived as possible. The merges to the master branch are done through a pull request. The peer reviewed pull requests work as the first quality gate that lets the code enter the whole CI pipeline and provides the developer with feedback on the work.

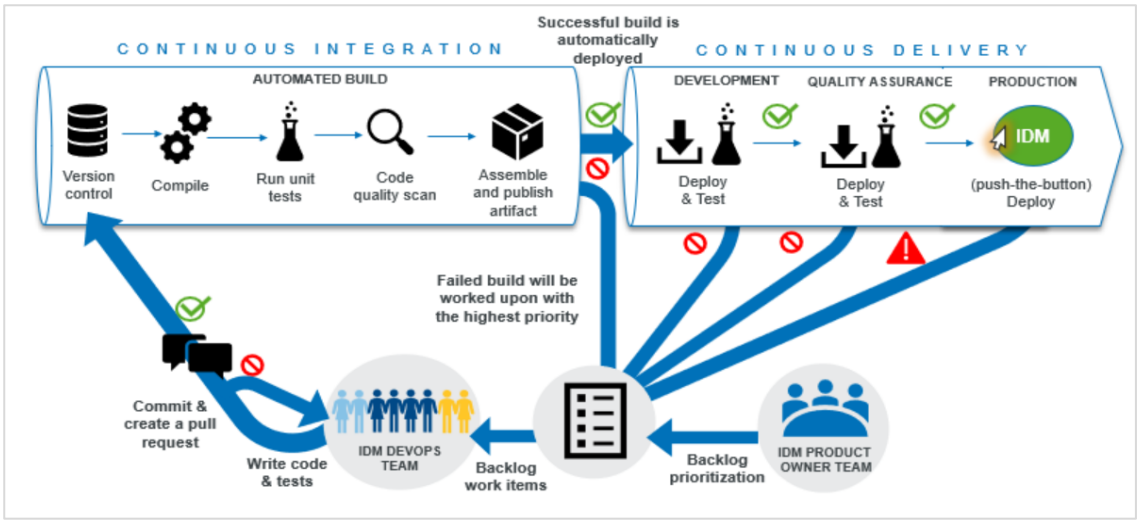

Figure 17 Artifact 2: IDM deployment pipeline (CI/CD)

The build process will be triggered automatically from each new check-in of the code to the version control (or alternatively, based on pre-defined schedule). The build will be ran using **Apache Ant**, which is a task-oriented build tool for Java applications. During the build, the code is automatically compiled and tested using unit test tool **Junit.** Dependency check tool **OWASP** and code quality scanner **SonarQube** will be integrated to the build to find code issues, bugs and vulnerabilities. Eventually the code is assembled and published. The successful build artifact will be published to the artifact repository in Azure DevOps. If the build does not pass the unit tests or defects have been found in the code quality scan, the build will fail and automatically create a bug to the backlog. The bug can be automatically assigned to the person who triggered the build; this way the bug will be solved by the person who committed the change and who still has the logic in fresh memory.

All the development work is deployed to the servers hosted in cloud. The successful build from the CI pipeline will trigger the CD pipeline, which will fetch the build artifact (.war file) from the artifact repository and deploy it to the servers in development environment. Successful tests in the development environment will trigger again the CD pipeline that will fetch the build artifact for QA environment and deploy it forward to the QA servers. The testing is done in small increments for every change committed to version control both in development and QA environment, before it is finally deployed to the production environment, wherein the changes will be in general availability. The regression tests in QA can be automated using **Robot Framework** and a **Selenium -tool**.

The QA environment will reflect the actual production environment as much as possible with integrations to the master data source system (HR -system) test environment and AD test environment. The performance and integration tests in QA will be done with authentic data volumes, which allows testing to be done with as realistic setup as possible. Finally, if all the automated tests pass also in the QA environment, the build can be deployed to the production environment where it will be in general availability to all Company X's users and applications. However, considering how mission critical the IDM is for the organization's overall ecosystem, the builds can be deployed into production manually after the build has passed all the tests in QA and the PO team together approves the changes.

6.2.4 Waterfall -based application onboarding process

All applications in Company X need to be onboarded to the IDM system to align with the company's information security policy; they must have authorizations properly lifecycle managed, and authentication arranged according to security guidelines. IDM system enables easier access rights provisioning and access certification to the applications by automating the processes. Applications can be onboarded to the IDM system by using three alternative provisioning strategies; automated provisioning with connectors directly to the application, automated pull provisioning using AD groups or manual provisioning with email or ticket to ticketing system. The automated pull provisioning using AD groups and manual provisioning are the more cost-effective, preferred provisioning strategies.

The application onboarding process is typically long process comprised of multiple manual steps such as info sessions and filling in forms (Figure 18). The effort required does not make the onboarding to IDM appealing to the application teams as the process is exhausting and requires dedication of time and resources both from the customer application team and from the IAM team. Eventually, many application teams refuse to onboard their application. The initial waterfall -based onboarding process starts with the

application onboarding kick-off session. After, IAM team will schedule an interview with the application team to gather initial requirements for the onboarding, such as the details of the needed roles, related entitlements and the approvers of the roles. Later, IAM team holds a status call to the applications in where they must commit to a 'basket'. These three steps introduce the first friction point to the process during which application team may drop off from the process already. Time is used for committing to sessions and interviews before actually gathering the needed requirements from the application team.

Each application onboarding basket includes approximately 20 applications that will be deployed to the IDM at once. There are several parallel streams of these baskets. When an application team has committed to the basket's schedule, the process starts with two info sessions held by IAM team, in which the application teams are instructed to fill in an excel file template that is used to collect the data. These files will be further converted to comma-separated value files (.csv) which can be imported to the IDM system. These data collection sessions introduce the second friction point due to the manual work that is prone to error and misunderstanding. The data collection excel is full of interpretation and not prepared with user friendliness in mind. The info sessions will follow with a third friction point which is the manual data cleaning, validation, import, testing and fixing required before the data can be imported to IDM. After the deployment, the applications are provided two weeks of hyper care. During hyper care period, the IAM support team will tackle all incidents and problems that appear in the production. The whole cycle for each basket requires up to six months of dedication from the teams.

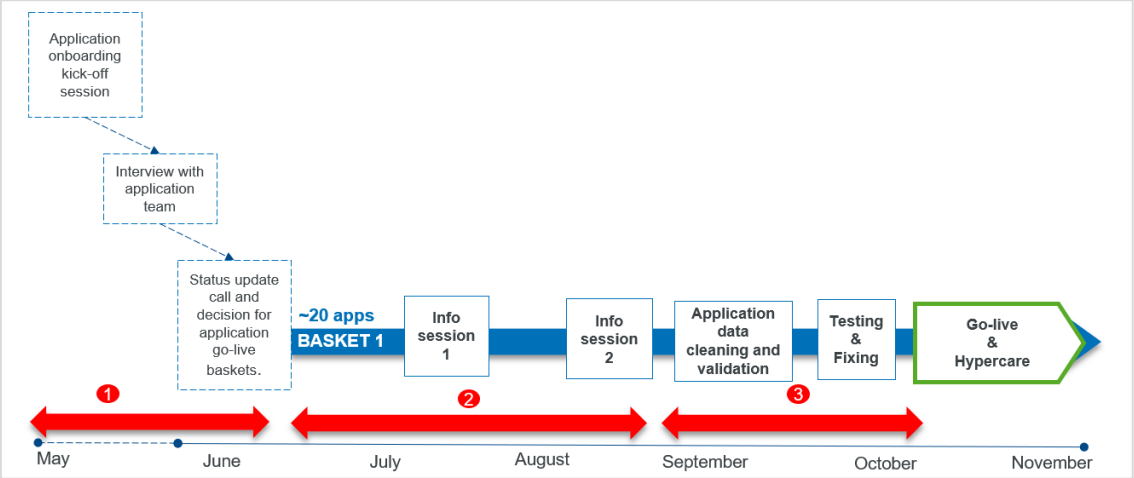

Figure 18 Application onboarding in the waterfall model

The question here is how DevOps practices could be adopted in the application onboarding to streamline the process in Company X. The target is to improve the application onboarding process to make it faster and easier so that the application teams can onboard their applications to IDM in a self-enabled way. With the help of DevOps principles and

practices, IAM team could provide application onboarding to IDM as a continuous service.

6.2.5 Application onboarding as a continuous service

To start with, the baskets containing 20 applications that are onboarded at once must be broken down into single item flow. Having each application as a smaller work item their status is easily trackable and the work becomes more transparent. If one application has a simple authorization model that does not require refinement or further support, the application team could easily fill in the requirement form on the same day, be processed in the IDM pipeline and be ready for deployment. In addition, if the deployment-ready application is waiting in the basket, the business value cannot be derived from the IDM as efficiently as if the application was deployed on the same day. Moreover, the data provided by applications gets outdated almost instantly as they have provided it to the IDM team. The faster the applications are onboarded to IDM, the faster they have the authorizations automated and lifecycle managed from one central IDM system. In other words, by speeding up the process, IAM team can increase information security, user experience and operational excellence in a faster, more efficient way.

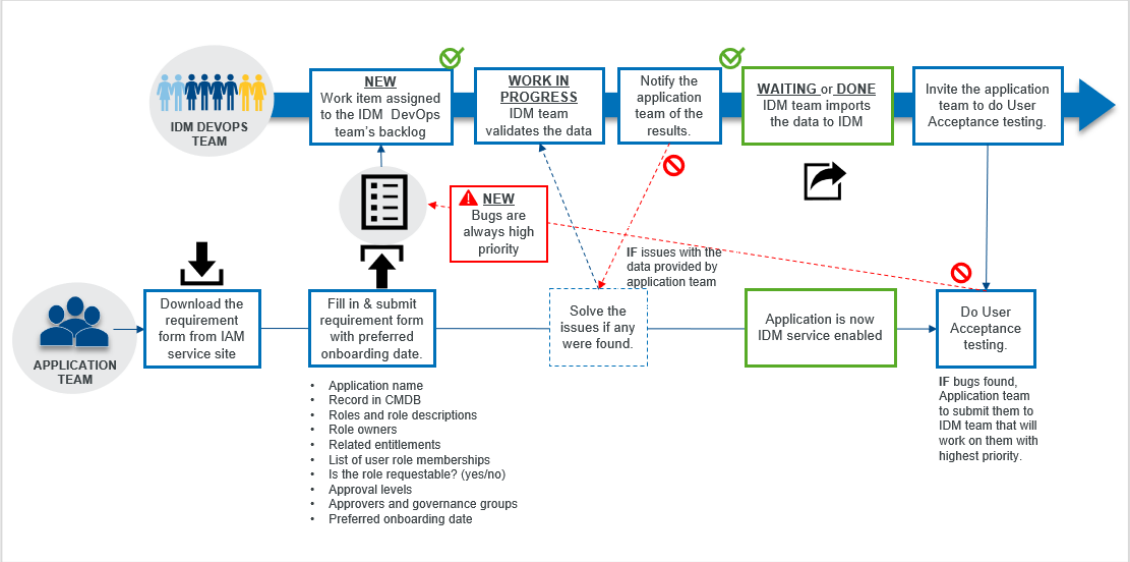

Figure 19 Artifact 3: Application onboarding as a continuous service

The continuous service is started by application team filling in the requirement form which should be as simple as possible in a way that the application team can fill it in by themselves without consultancy and the IDM DevOps team is able to receive all data needed to import to the IDM. This requirement form would collect data such as the name

and ID of the application, all the roles that will be permitted to access the application and the role owners, the entitlements that will be given with the roles (e.g. security group in AD that allows the user with that role to only read data inside the application), the list of existing role memberships, define approval levels for those roles, and other needed information. Most importantly, for better customer experience, the application can set their preferred onboarding date such as “As soon as possible” or “Next week”, and follow-up on the progress of their application onboarding.

To provide this service in a continuous matter, the requirement form should be easily available online and easy to hand over to the DevOps team. Also, the team should be working on requirements, validating and fixing the data provided by the applications, importing the data to IDM and providing/receiving feedback continuously instead of executing tasks stage by stage. Whenever data requires validation, the DevOps team will enforce principle of efficient feedback and collaborate with the application team to bring the work forward in the deployment pipeline.

After all the data has been imported to IDM, roles, entitlements and new AD groups have been created, the application is ready to authorize its users using the automated, centralized provisioning via IDM. If required, the IDM DevOps team will send an automatic invite to the application team to conduct User Acceptance Testing. In case any errors are found in the application’s access provisioning, a bug will be raised to the IDM DevOps team’s backlog and they will be worked upon with highest priority. As the team has the automated deployment pipeline in use, the bug fixes are visible in production already during the same day.

7 EVALUATION

7.1 Artifact evaluation

This chapter evaluates and discusses how the previously introduced design artifacts 1, 2 and 3 (Figure 16, Figure 17, Figure 19) function and perform in the application domain. The evaluation is made based on observations and discussions with the key people in the application domain and the IAM technology vendor. The observations and discussions were made during the iterations, and especially the retrospective meetings at the end of each iteration provided valuable inputs to the evaluation. The conclusion made after the design construction and evaluation process is that DevOps cannot be pre-planned and easily applied in the Identity & Access Management area as it is. In fact, adopting DevOps in Identity and Access Management requires continuous learning and design work to match the practices and tools to fit with the technology, people and organization. The adoption of Continuous Integration, Continuous Delivery and Continuous Feedback practices were largely affected by the chosen IDM technology (SailPoint IIQ) and DevOps tools, the developers' and testers' skills and background, and the organizational processes in which the Identity & Access Management team functions. The following subsections will describe in more detail the results of the artifact evaluation during the case study and the special considerations that had to be made regarding the DevOps adoption in all four previously identified main problem areas.

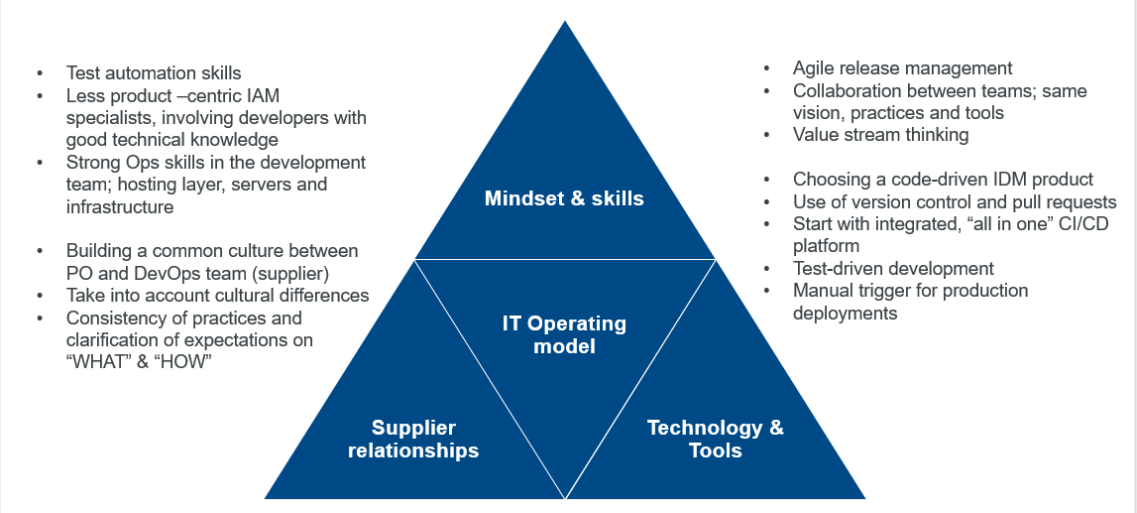

Figure 20 Summary of the success factors for DevOps adoption in IAM using the identified four main problem areas

7.1.1 Technology & tools

The use of version control system was found to be the corner stone for adopting Continuous Integration in the IDM delivery. To start with, the use of version control was found to be at immature level in the IDM development team. Without structured use of version control the team was not be able to track the work, deliver often, obtain quality or restore changes. Based on discussions with the vendor and IAM experts in the application domain, there are not many IDM solutions in the market (if customer IAM is opted out) that would be code driven; allow control over versioning or the build process. The IIQ product that was chosen was code-driven, and that became a critical success factor for the DevOps adoption.

Having trainings and sessions about correct way of committing changes, and assignment of a technical architect to approve the developers' pull requests was important to safeguard the quality. However, it was found that configuring the IIQ was after all faster and more convenient to do from the user interface in certain cases, instead of directly to the version control. In that case, the UI-configured object had to be exported from the application and committed to version control, and then again deployed to the environments. If the pipeline was triggered to ran in the continuous deployment mode, the UI-based configuration would create a bottleneck to the development and become an unsustainable practice.

Automating the build process introduced difficulty in having all pieces automated, such as configuration of environment specific encryption keys securely as part of the build. However, according to the developers the automated build and deployment processes made their work easier and faster as they always had the latest changes available for validation in the development and test environments. Having the automated unit tests running in the CI pipeline was found to be more difficult with the IIQ product, than expected. In IIQ, the customization files are written in pure Java programming language. However, most of the changes committed by the developers were in the configuration files that are written in BeanShell, which is a mixture of Java programming and scripting language. Therefore, the same unit testing process and tools were not possible to apply for both configuration and customization code. Although the pure Java files could be unit tested automatically as part of the build, some considerations had to be made to have unit tests ran for all changes.

The IIQ is shipped with a unit test framework that is based on running unit tests on BeanShell manually from user interface, which does not support DevOps practices. The vendor also offers a unit testing tool (JUnitHelper) as part of their deployment kit that supports testing the BeanShell. The problem with JUnitHelper turned out to be that the tool requires a running IIQ instance to run the unit tests. This means that the unit tests

would need to be moved to the CD pipeline by deploying the build first to the development environment and run the unit tests in there. Based on the principle of building efficient feedback loops, ‘shift left’ testing is recommended practice for DevOps. Therefore, a customized unit test tool that simulates IIQ instance as part of the build, had to be coded specifically for that purpose. Automated regression testing was found to be a success factor for speeding up the process in between the automated builds and deployments. The test automation brought the team members together to do test-driven development. The test-driven development approach made the team to think more about the technical implementation of features before writing the code. For example, the test-driven approach forced the team to think how to write the code in a way that it can be easily tested automatically and that the tests can be used repeatedly.

Continuous Delivery pipeline was built for IIQ by using scripts that fetch the build artifact from Azure DevOps artifacts repository and copy and deploy the build first to the servers in development, QA and production environment. Deployments to the development environment were triggered automatically after each finished, automated build. Moving forward to QA, having a developer trigger the automated deployment pipeline manually after validating the changes in development environment, was a preferred way of working. Using manual trigger, the deployment would not interfere with the scheduled tasks that were running on the QA servers. In IIQ, there are refresh and correlation tasks that are running to get the latest data changes from the integrated systems, for example to refresh the identity data. In deployments, all the servers are restarted, which would interrupt these tasks. However, although the deployments were not done all the time, the pipeline was configured for Continuous Delivery of artifacts that could be deployed on a suitable moment. Continuous Delivery enabled faster validation of the code and detection of errors. Also, the ability to deploy, restore the changes and re-deploy enabled the team to work in more efficient iterations.

Overall, the big success factor turned out to be the CI/CD platform that was chosen (Azure DevOps) to be used. By using “all in one” platform for starting to adopt the new DevOps tools and practices was easier for team that had no earlier experience of DevOps. Also, by having existing integrations in the platform with both open source and commercial tools provided with options and reduced the need for building and maintaining integration in between tools, such as integration of build/test tools with CI server and deployment tools/automated tests with the CD server.

When it comes to application onboarding as a continuous service, the requirement gathering was found to be an issue. The IAM team believed and had experienced before that the applications alone were not enough informed about IAM and the technologies behind it to provide inputs to IAM team with regards to their preferred authorization models. For example, not all applications can map entitlements to roles and understand how to authorize their users using AD groups, and thus would not fell at ease when submitting

requirements. Also the technology vendors noted that the biggest problem when trying to speed up the application onboarding is the immense data collection and structuring of the data received from the applications. Regardless of that, IAM would benefit from adopting DevOps in the application onboarding process as even the few applications that do have easier authorization models could be onboarded faster. Also, the team would save time and efforts dedicating their time to consulting the applications with more difficult authorization models.

7.1.2 Mindset and skills

The DevOps team was formed by bringing people from development, testing and operations together to continuously deliver a product with built-in quality. What is typical to the field of Identity & Access Management, is that the development team is composed of IAM specialists who have experience mainly of configuring the IDM product from the user interface. These IAM specialists often know the product functionalities but lack experience of the high-quality development and testing practices. Therefore, the original level of agile/DevOps knowledge in the IDM implementation team was low.

The IDM Product Owner team had already built certain "DevOps vision" and developed expectations on the DevOps team without understanding how small steps were actually required with people who had such a "narrow" skill set. When forming the DevOps mode of IDM delivery, it was found important to collect a team of people who have a good understanding on the overall development techniques and underlying technologies instead of a team of specialists only deeply focused on the IDM product. What seemed to be more efficient is to dedicate resources and time to the team members who maybe lacked some of the subject matter expertise, but instead had strong technical background with the tools, and ability to understand end-to-end processes. The most critical success factor was to have people with strong "Ops skills", e.g. with hosting layer and specifically Linux environments, included in the development team. These people turned out to be of most value when building the automated deployment pipeline, scripting and troubleshooting server -side issues.

7.1.3 Suppliers

Also the cultural differences between suppliers and IAM team affected the DevOps adoption in the IAM area. The members of the DevOps team were from a cultural background in which the employees are accustomed to the stronger hierarchy and 'command-and-control' type of culture in the work place, while the PO team consisted of people who

were already very comfortable with high-trust, low hierarchy culture at the work place. Adopting DevOps culture was the opposite of the DevOps team's original work culture and the learning was that the PO team cannot expect the development team to immediately adopt opposing cultural behaviors. It was also critical for PO team to be clear and consistent when stating their expectations to the DevOps team. As raised in the interviews held in Company X, when involving suppliers into the project, it is more important to define how the work needs to be delivered instead of only what needs to be delivered. PO team had to take a different approach and provide more support and guidance in the start and invest in building long-lasting respectful relationship with the DevOps team members, so that each member of the team feels comfortable and shares mutual trust with one another, but also with the PO team.

7.1.4 IT operating model

As it was raised by the interviews in Company X's IT unit, some of the existing processes and organizational structures defined in IT Operating model do not support agility and DevOps. Not only Identity & Access Management team, but also the other teams in the company mentioned that they are tackling with many challenges related to release management, project/portfolio management and enterprise architecture. Many friction points were raised that could help Company X to better take advantage of the efficiency and speed that DevOps could provide.

For example, the existing manual release management process in Company X is not agile. While IAM team was running the project in DevOps mode, there were still slow organizational processes that had to be followed. For example, the IAM Solution Manager had to dedicate his time to manual updates to change management tools where he would need to create change requests for every item that the DevOps team intends to deploy to production. Also, the existing change management tool does not align with the DevOps mode of working in which the changes are deployed efficiently in a continuous manner. Instead of having the changes approved by someone situated in separate unit, the change approval process should be brought to as close to the person who knows what the changes are about.

There were also some dependencies with other teams' slow and heavy processes that slowed down the DevOps way of working in IAM team. Firstly, the cloud resources were managed and delivered by the Cloud team whose daily operations are handled by the cloud service provider. The delivery of the initial environment setups took up to two months and required filling up templates and having many conversations with the cloud service provider. During the project, if extra capacity was needed or something was wrong

with the servers in development/QA or production environments, the DevOps team would need to raise a ticket to the cloud service provider and wait for approvals and queues.

Second example from the project is the Active Directory refresh activity. The AD is owned and managed by another team at the supplier side. The identity data in the test environment's AD, to which IIQ was connected to, needed to be refreshed during the project to have as authentic setup as possible in QA environment. Eventually it took one and half weeks to refresh the AD as the third-party vendor required change requests, approvals, and the communication was slow. To succeed in the DevOps adoption, the IAM team needs to collaborate with multiple different teams that would ideally share the same vision, practices and tools. In order to get there, the teams should start thinking about the whole value stream and what is their own role in the organizations' DevOps journey.

8 CONCLUSIONS

8.1 Conclusions

Earlier studies state that DevOps is not a process that can be taken as it is and implement, and therefore adoption of DevOps is not easy to do. In addition, executives often have overly positive view on the progress and impacts of DevOps adoption in their organizations (State of DevOps 2018). There is a lack of research on how DevOps is adopted in specific areas of IT, although it has been concluded that DevOps is an artifact that always needs to be adapted to its environment. This research aimed to answer to the main research question "***How to adopt DevOps principles, practices and tools?"*** by conducting a design science research in Identity and Access Management. The main research question was broken down into smaller sub questions that helped to construct and evaluate the design artifacts for DevOps adoption in IAM. IAM is an area of IT that is often not associated with the latest development practices, and no existing research is found regarding DevOps adoption in IAM. Adopting DevOps in IAM could help organizations to improve their operational excellence, security risk management and user experience. The design science research was conducted using a case study approach, wherein the artifacts were observed and analyzed in the problem domain.

First, the DevOps analysis was made in the case company's IT unit to gain deeper understanding on what are the challenges of DevOps adoption at an organizational level. Four main challenges were raised: 1. IT Operating model, 2. Supplier relationships, 3. Technology and tools, and 4. Mindset and skills. It was found that more than half of the participants did not yet have a good knowledge level on what is DevOps and that the teams need centralized support for DevOps adoption from the organization.

After the initial analysis, three design artifacts were built to describe how to adopt DevOps principles, practices and tools in the IAM team's IDM service delivery and application onboarding process. The first artifact (Figure 16) illustrates how the IAM team should re-construct the siloed teams into cross-functional teams and change the way of delivering the IDM system from waterfall to agile. The model is based on DevOps principles that emphasize continuous delivery of value. The second artifact (Figure 17) illustrates the process and construction of the automated deployment pipeline with DevOps practices and tools for IDM system development to optimize the workflow. The third constructed artifact (Figure 19) describes the process of onboarding applications to the IDM following DevOps principles.

The design artifacts were evaluated by considering the identified four main problem areas of DevOps adoption and the success factors for DevOps adoption in IAM. This research strengthened the conclusions made from previous studies that DevOps cannot be

pre-planned and adopted in IAM as it is, but instead requires several iterations and continuous design work. The final results of the evaluation raised the importance of choosing a code-driven IDM technology that supports use of version control and the ability to manage the changes. The 'all in one' CI/CD platform turned out to be an important success factor for further adoption of DevOps tools and practices as it reduced the need to integrate all tools separately and maintain the integrations. It was also found that IDM specialists often lack knowledge of development best practices and have very product-centric skills. In DevOps adoption, it became important to find the people who have end-to-end understanding and excellent development/operations skills instead of having traditional IAM specialists in the team who know the product inside out. Having people with strong 'Ops' skills in the development team was found to be a critical success factor, as these people had skills to automate the deployment pipeline and troubleshoot issues on the servers.

This research showed brought up the challenges that should be overcome in a large organization for DevOps adoption, and how DevOps principles, practices and tools can be adopted in the field of IAM. The discussions that were held at the evaluation phase with the IAM technology vendor also revealed that they had never seen such maturity of automation in IDM implementations, referring to this case as "the cutting edge of DevOps in IAM". The technology vendor -side also showed interested to gain access to the technical details and share the learnings of the DevOps adoption forward to the other practitioners in the IAM community.

In the case company (Company X), the DevOps adoption in the IAM area is seen as a strong showcase of DevOps adoption. The leaders from different areas of the company see that this case from IAM area provides valuable insights and concrete help to the other teams in their DevOps journey. The analysis showed that the teams from different areas in the IT unit find it challenging to start DevOps adoption from scratch, stating that the expertise and lessons learned from DevOps adoption should be shared with the other areas so that they could develop their own adoption model. The results also highlight the need to improve the IT operating model, relationships with the suppliers, technology and tool base, and mindset and skills to support IT organization's transition to DevOps. However, in addition to the bottom-up motivation, the successful DevOps adoption would require strong top-down sponsorship. In addition to the organizational change management efforts, the top management could support the development of DevOps practices by allocating resources to centralize the skills and capabilities in a way that the framework can be scaled in the organization.

8.2 Limitations and future study

The design artifact evaluation of this research was restricted by the time constraints of the case study and therefore the functionality of certain aspects of the design artifacts could not be comprehensively evaluated. Firstly, the unit test automation was designed and implemented in the sandbox environment but not included to the automated DevOps pipeline because the maturity of the team to write unit tests was not at required level. Secondly, the duration of the case study was not long enough to have experimented the Artifact 3 in the problem domain. The evaluation of the Artifact 3 was thus only based on discussions with the experts.

This study could be extended to measure the business impacts of Artifact 3 and the customer experience of application onboarding as a continuous service. How onboarding to IDM could be made simpler to the applications so that the actual onboarding could be automated to larger extent? Additionally, to expand the research on adopting DevOps in IAM, the containerization solutions and serverless computing could be further studied. How containers can be used in IAM solutions to speed up the provisioning of IAM environments and roll backs in critical situations? Furthermore, as this study acknowledged the challenges to provide IAM services in a continuous mode in an organization where other teams are not aligned and share the same DevOps vision and practices; what are the challenges in an environment where other teams have adopted DevOps as well? In this research, DevOps way of working was slowed down by heavy waterfall-based release management process of the company. An interesting question for future research could be that how should the release management be arranged around IAM solutions in large companies when these services are offered following DevOps principles, practices and tools?

As concluded earlier, DevOps adoption is not something that can be planned and repeated from one solution to another. DevOps combines practices and tools that may work for one solution but some of them may not always be applicable. However, DevOps is an artifact that can be adopted to help teams to accelerate the delivery of value and improve performance. Thus, this research invites new study that aims to explore and design frameworks on how to adopt DevOps in different areas of IT delivery.

9 REFERENCES

Aken, J. E. V. 2004. Management research based on the paradigm of the design sciences: the quest for field-tested and grounded technological rules. *Journal of management studies*, 41(2), 219–246.

Artac, M., Borovssak, T., Di Nitto, E., Guerriero, M., & Tamburri, D. A. 2017. DevOps: introducing infrastructure-as-code. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C).*497–498. IEEE.

Balaji, S., & Murugaiyan, M. S. 2012. Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. *International Journal of Information Technology and Business Management*. 2(1). 26–30.

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

Bass, L., Weber, I., & Zhu, L. 2015. *DevOps: A software architect's perspective.* Addison-Wesley Professional.

Beck, K. 1999. Embracing change with extreme programming. *Computer*. 10, 70–77.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., & Kern, J. 2001. Manifesto for agile software development.

Beller, M., Gousios, G., & Zaidman, A. 2017. Oops, my tests broke the build: An explorative analysis of Travis CI with GitHub. *In 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 356–367. IEEE.

Bertino, E., & Takahashi, K. 2010. *Identity management: Concepts, technologies, and systems*. Artech House.

Bird, J. 2016. *DevOpsSec: securing software through continuous delivery.* O'Reilly Media Inc. <https://learning.oreilly.com/library/view/devopssec/9781491971413/> retrieved in 25th February 2019.

Boehm, B. 2006, May. A view of 20th and 21st century software engineering. In *Proceedings of the 28th international conference on Software engineering*. 12–29. ACM.

Borgenholt, G., Begnum, K., Engelstad, P. 2013. Audition: A DevOps-oriented Service Optimization and Testing Framework for Cloud Environments. In: *Conference of Norsk informatikkonferanse (NIK).* 146–157. Akademika Publishing, Trondheim.

Boyce, C., & Neale, P. 2006. Conducting in-depth interviews: A guide for designing and conducting in-depth interviews for evaluation input. < http://dmeforpeace.org/sites/default/files/Boyce_In%20Depth%20Interviews.pdf> retrieved in 11th February 2019.

Cao, L., Mohan, K., Xu, P., & Ramesh, B. 2004. How extreme does extreme programming have to be? Adapting XP practices to large-scale projects. *In 37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the IEEE.* 1–10.

Chef Blog. 2013, January 22nd. *Prezi Makes Your Ideas Matter w/Hosted Chef* by Welch, L. <https://blog.chef.io/2013/01/22/prezi-makes-your-ideas-matter-whosted-chef/> retrieved 27th March 2019.

Cohen, D., Lindvall, M., & Costa, P. 2004. An introduction to agile methods. Advances in computers, 62(03), 1–66.

Conway, M. E. 1968. How do committees invent. *Datamation*. 14(4). 28–31.

Cukier, D. 2013, October. DevOps patterns to scale web applications using cloud services. *In Proceedings of the 2013 companion publication for conference on Systems, programming, & applications: software for humanity* 143–152. ACM.

Dyck, A., Penners, R., & Lichter, H. 2015. Towards definitions for release engineering and DevOps. *In 2015 IEEE/ACM 3rd International Workshop on Release Engineering* (3-3). IEEE.

Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. 2016. DevOps. *IEEE Software*, 33(3). 94–100.

Elshamy, A., & Elssamadisy, A. 2007. Applying agile to large projects: new agile software development practices for large projects. *In International Conference on Extreme Programming and Agile Processes in Software Engineering*. 46–53. Springer, Berlin, Heidelberg.

Erich, F. M. A., Amrit, C., & Daneva, M. 2017. A qualitative study of DevOps usage

in practice. *Journal of software: Evolution and Process*. 29(6). 1–14.

Erich, F., Amrit, C., & Daneva, M. 2014. Report: Devops literature review. *University of Twente*. <https://www.researchgate.net/profile/Chintan_Amrit/publication/267330992_Report_DevOps_Literature_Review/links/544ba33f0cf2bcc9b1d6bd8a.pdf> retrieved 25th February 2019.

Fitzgerald, B., & Stol, K. J. 2014. Continuous software engineering and beyond: trends and challenges. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*. 1–9. ACM.

Forbes.com. The World's Most Innovative Companies. <https://www.forbes.com/innovative-companies/#1d6b31141d65> retrieved in 20th February 2019.

Forsgren, N., Humble, J., & Kim, G. 2018. *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations.* IT Revolution.

Freeform Dynamics. Assembling the DevOps Jigsaw. 2015. <https://www.ca.com/content/dam/ca/us/files/msf-hub-assets/research-assets/assembling-the-devops-jigsaw.pdf>, retrieved 2nd January 2019.

Gall, M., & Pigni, F. 2018. Leveraging DevOps for mission critical software. In *Emergent Research Forum*.

Gartner IT Glossary: Bimodal IT. <https://www.gartner.com/it-glossary/bimodal> retrieved in 17th Janurary 2019.

Gartner. 2018. Report: Magic Quadrant for Software Test Automation.

Ghantous, G. B., and Gill, A. 2017. DevOps: Concepts, Practices, Tools, Benefits and Challenges. In *PACIS 2017 Proceedings*.

Gonzalez, D. 2017. *Implementing Modern DevOps.* Packt Publishing, Birmingham.

Gregor, S., & Hevner, A. R. 2013. Positioning and presenting design science research for maximum impact. *MIS quarterly*, 37(2).

Gregor, S. 2006. The nature of theory in information systems. *MIS quarterly*, 611–642.

Gruver, G. 2016. *Starting and Scaling DevOps in the Enterprise*. < https://www.microfocus.com/media/ebook/Starting-and-Scaling-DevOps-in-the-Enterprise.pdf> retrieved in 21st March 2019.

Gruver, G., & Mouser, T. 2015. *Leading the transformation: Applying agile and devops principles at scale.* IT Revolution.

Gruver, G., Willis, J. & Kim, G. 2015. *Mythbusting DevOps in the Enterprise*. IT Revolution.

Haffke, I., Kalgovas, B., & Benlian, A. 2017. Options for Transforming the IT Function Using Bimodal IT. *MIS Quarterly Executive*, 16(2).

Hering, M. (2018). *DevOps for the Modern Enterprise: Winning Practices to Transform Legacy IT Organizations*. IT Revolution.

Hevner, A., & Chatterjee, S. 2010. Design science research in information systems. In Design research in information systems. 9–22. Springer, Boston, MA.

Hevner, A., March, S., Park, J. & Ram, S. 2004. Design Science in Information Systems research. *MIS Quarterly.* 28(1), 75–105.

Highsmith, J., & Cockburn, A. 2001. Agile software development: The business of innovation. Computer, 34(9), 120–127.

Horlach, B., Drews, P., Schirmer, I., & Böhmann, T. 2017, January. Increasing the agility of IT delivery: five types of bimodal IT organization. In *Proceedings of the 50th Hawaii International Conference on System Sciences*.

Hsu, T. 2018. *Hands-on Security in DevOps: ensure continuous security, deployment and delivery with DevSecOps.* Packt Publishing Ltd.

Humble, J., & Farley, D. 2010. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education.

Ikonen, M., Kettunen, P., Oza, N., & Abrahamsson, P. (2010). Exploring the sources of waste in kanban software development projects. In *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications*. 376–381. IEEE.

Jacobs, F. R. 2007. Enterprise resource planning (ERP)—A brief history. Journal of Operations Management, 25(2), 357–363.

Kaur, K., & Jajoo, A. 2015. Applying agile methodologies in industry projects: Benefits and challenges. In *Computing Communication Control and Automation (ICCUBEA), 2015 International Conference*. 832–836. IEEE.

Kim, G., Humble, J., Debois, P., & Willis, J. 2016. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution.

Kneuper, R. 2017. Sixty Years of Software Development Life Cycle Models. IEEE Annals of the History of Computing, 39(3), 41–54.

Laukkarinen, T., Kuusinen, K., & Mikkonen, T. 2018. Regulated software meets DevOps. *Information and Software Technology*, 97, 176–178.

Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D. & Kahkonen, T. 2004. Agile software development in large organizations. *Computer.* 37(12). 26–34.

Lwakatare, L. E., Kuvaja, P., and Oivo, M. 2015. Dimensions of DevOps. In *Agile Processes in Software Engineering and Extreme Programming* (Vol. 212), C. Lassenius, T. Dingsøyr, and M. Paasivaara (eds.), Cham: Springer International Publishing. 212–217.

Machiraju, S., & Gaurav, S. 2018. *DevOps for Azure Applications.* Apress.

March, S., & Smith, G. 1995. Design and natural science research on information technology. *Decision support systems*, 15(4), 251–266.

Meyer, M. 2014. Continuous integration and its tools. *IEEE software*, 31(3), 14–16.

Nerur, S., Mahapatra, R., & Mangalaraj, G. 2005. Challenges of migrating to agile methodologies. Communications of the ACM, 48(5), 72–78.

Ravichandran, A., Taylor, K., & Waterhouse, P. 2016. *DevOps for Digital Leaders.* Apress, Berkeley, CA.

Reed, L. (2015, June 22nd). *The Simple Math of DevOps.* <https://devops.com/the-simple-math-of-devops/>, retrieved in 20th March 2019.

Rolland, K. H. 2016, May. Scaling Across Knowledge Boundaries: A Case Study Of A Large-Scale Agile Software Development Project. In *Proceedings of the Scientific Workshop Proceedings of XP2016*. ACM.

Royce, W. 1970. Managing the Development of Large Software Systems. IEEE Wescon. 1–9.

Sedano, T., Ralph, P., & Péraire, C. 2017, May. Software development waste. In *Proceedings of the 39th International Conference on Software Engineering* (130–140). IEEE Press.

Sharma, S. 2017. *The DevOps Adoption Playbook: A Guide to Adopting DevOps in a Multi-Speed IT Enterprise*. John Wiley & Sons.

Simon, H. A. 1996. *The sciences of the artificial (*3rd ed). MIT press. Cambridge, MA.

Soni, M. 2015. End to end automation on cloud with build pipeline: the case for DevOps in insurance industry, continuous integration, continuous testing, and continuous delivery. *In 2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)* (85–89). IEEE.

Stake, R. E. 1995. *The art of case study research*. Sage.

State of DevOps Report 2018. Puppet labs.

State of DevOps Report 2014. Puppet labs.

Stober, T., & Hansmann, U. 2010. *Best Practices for Large Software Development Projects*. Springer.

Turner III, D. W. 2010. Qualitative interview design: A practical guide for novice investigators. *The qualitative report.* 15(3). 754–760.

Venable, J., Pries-Heje, J., & Baskerville, R. 2012. A comprehensive framework for evaluation in design science research. *In International Conference on Design Science Research in Information Systems*. 423–438. Springer, Berlin, Heidelberg.

Verona, J. 2016. *Practical DevOps.* Packt Publishing, Birmingham.

Virmani, M. 2015, May. Understanding DevOps & bridging the gap from continuous integration to continuous delivery. In *Innovative Computing Technology (INTECH), 2015 Fifth International Conference.* 78–82. IEEE.

Wengraf, T. (2001). *Qualitative research interviewing: Biographic narrative and semi-structured methods.* Sage.

Wettinger, J., Breitenbücher, U., Kopp, O., & Leymann, F. 2016. Streamlining DevOps automation for Cloud applications using TOSCA as standardized metamodel. *Future Generation Computer Systems*, 56, 317–332.

Wiedemann, A. 2017. A New Form of Collaboration in IT Teams - Exploring the DevOps Phenomenon. In *PACIS 2017 Proceedings*.

Womack, J.P., Jones D.T., & Ross D. 1990. *The Machine that Changed the World*, Rawson Associates. New York, NY.

Zimmerman, J., & Forlizzi, J. (2008). The role of design artifacts in design theory construction. *Artifact: Journal of Design Practice*, 2(1), 41–45.

Zimmerman, J., Evenson, S., & Forlizzi, J. (2004). Discovering and Extracting Knowledge in the Design Project. In *Proceedings of FutureGround*. Design Research Society.