Universiteit Leiden

ICT in Business and the Public Sector

DevOps Unravelled: A Study on the Effects of Practices and Technologies on Organisational Performance

Name: Robert Blinde
Student-no: 1370162

Date: 15/04/2022

1st supervisor: Dr. C.J. Stettina MSc
2nd supervisor: T.D. Offerman MSc

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

DevOps Unravelled: A Study on the Effects of Practices and Technologies on Organisational Performance

Robert Blinde

Leiden Institute of Advanced Computer Science (LIACS)
P.O. Box 9512, 2300 RA Leiden, The Netherlands

April 15, 2022

Abstract

After waterfall and Agile, DevOps is the latest trend in software delivery. Evolved from the Agile mindset, DevOps aims to shorten lead times and increase the quality of software through continuous delivery [1], making organisations more flexible to changes in the environment [2].

Depending who you ask, DevOps can be described in different ways: a role, a group of people "doing" DevOps, or a mindset, much like Agile. In fact, DevOps has been difficult to define [3] and thus many definitions of DevOps exist [4, 5]. Similarities between the definitions include bridging the gap between Dev and Ops teams, adhering to DevOps practices, and improving communication and collaboration. Using the CALMS framework, DevOps and its tools and practices will be explained in detail.

Research has shown that adopting DevOps can be fruitful; after correct implementation, numerous benefits can be expected. Some of these include: shorter cycle times [6–9], more frequent releases [10, 11], increase in quality [6, 7, 9], and improved collaboration and communication [7, 12]. Correct implementation, however, can be hindered by different challenges. DevOps can be hard to define [3, 7, 8], requires a shift in organisational culture [3, 7, 13–15], as well as approval from management [8, 14]. Combatting these challenges is key in the adoption process.

This research studies the effects of DevOps practices and technologies on organisational performance. Using a web-based survey, data was collected from 123 DevOps practitioners, consultants, and their managers. The survey included different sections: demographics, transformation, maturity, DevOps practices and tools, and impact and (software delivery) performance.

DevOps and Agile maturity are closely related. The Organisation and Culture layer of the DevOps maturity model had the highest average score. The required shift in organisational culture has thus successfully been combatted by a large number of respondents. Unplanned work generally decreases when maturity increases, as well as the time spent working on features. Maturity can be improved by adhering to three DevOps practices: 1) automated and continuous testing in development and staging

environments, 2) automated and continuous deployments, and 3) automated and continuous monitoring. Two of those practices are already commonly in use by practitioners. Those, and the most common one – everything stored as code and under version control – also have a positive impact on both software delivery performance and organisational performance. Popular DevOps tools include Azure DevOps, Jenkins, GitLab, and Bamboo. Bamboo and Datadog are solely used by high performers. They also tend to use more GitLab CI, but use Azure DevOps and Jenkins less often. Other technologies, such as containerisation and open-source software have shown to fuel DevOps.

After unravelling the mysteries around DevOps, its practices and tools, adhering to practices that improve lead time (e.g. continuous and automated deployments or everything as code and under version control) will have a positive impact on both software delivery performance and organisational performance.

Acknowledgements

This research was started after the initial inspiration of Tyron Offerman MSc. Our biweekly meetings were of incredible value to me, how efficicent they sometimes were. Your ability to respond quickly to any questions I had, allowed me to keep working without losing focus. Next, I would like to thank my first supervisor, dr. Christoph J. Stettina, for his extensive acedemic knowledge and feedback, as well as the opportunity to present a part of my results at the 2021 RTE Summit in Utrecht. Your constructive feedback has improved the overall quality of this research. Lastly, and most importantly, I would like to thank my parents and girlfriend for their continuous support and by helping me clear my mind while writing this thesis. Thank you all.

Contents

Chapter 1

Introduction

Nowadays, most aspects of our lives are in some way connected to software. We use software programmes and cloud applications at work, connect with like minded people using social media platforms, and consume the content of smartphone apps or streaming services (e.g. Spotify for music or Netflix for movies) when we want to be entertained. All these pieces of software have been carefully developed with a clear business goal in mind. Possible goals include: high user satisfaction, increase in sales, or reaching a certain number of users.

Not too long ago [1] software was developed using a traditional waterfall method. Characteristics of traditional (or linear) software development projects are: long lead times, inflexible requirements, and little involvement of clients throughout the process. Users could grow impatient while waiting for the next large update, or even disappointed when bugs are discovered [14]. Other problems of the waterfall method are [14]: 1) developers do not fully grasp the trade-offs of running a production environment, 2) while developers focus on creating new features, the operations team focusses on stability, and 3) stakeholders get feedback months after project approval, slowing down future iterations.

Today, with continuous changes in both market needs and technology, organisations cannot afford long lead times. The waterfall method is therefore being replaced by new mindsets and methodologies: Agile and Scrum. A 2018 survey by Stack Overflow shows that 85.4% of the respondents use Agile [16]; an increase of 8.5% from the previous year [17]. In 2017, 26.9% of the respondents used waterfall methods. In 2018, this decreased to 15.1%.

Agile's main focus lies on adaptability and dealing with changes in customer requirements [18]. The Agile mindset is used within development departments to increase collaboration with product management [19]. Within Scrum, feedback from end-users is frequently used to update and improve products or services. It welcomes changes throughout the entirety of the project and adds new features in short iterative cycles. To reach business goals and maintain the competitive edge, companies thus require to quickly react upon their customers' feedback. However, the introduction of this new methodology comes with technical, cultural, and organisational stresses [1].

While the focus lied on increasing communication and collaboration between development teams and clients, the operations team – in charge of deploying and maintaining the newest versions of the software – was "left out of the revolution" [19].

DevOps was introduced to overcome the issue of misalignment between developers and operators. Although Agile and DevOps share the practice of continuous delivery, DevOps is much more than that [18].

The DevOps approach is concerned with the problems organisations encounter when trying to deliver incremental software in short cycles. DevOps combines the practices used in both Software Development and IT Operations and can be divided in roughly three phases: build, deployment, and operations [13]. The main goal of DevOps is to shorten lead times and increase the quality of software through continuous delivery [1], making organisations more flexible to changes in the environment [2]. New features are released frequently and bugs are solved quickly [14].

DevOps involves a shift in culture towards "collaboration between development, quality assurance, and operations" [13]. Two major elements in the context of DevOps are cross-department collaboration and automation [5, 7, 15]. Monitoring of applications and resources is also a key factor for a successful implementation [20]. Many practices seen in DevOps are inspired by those found within Lean, Agile, or The Toyota Way [21]. The common approach is to reduce the Work in Progress (WIP) by reducing batch sizes. This means, for example, breaking down software products into different functionalities and ship them one at a time. DevOps is similar to continuous delivery and both are used in parallel. However, DevOps also focusses on management and the required organisational change to effectively develop products and services [21].

DevOps can be used in most cases of software delivery [13], requiring special configurations for both the environment and architecture. It allows organisations to plan and deliver software updates quickly. A good example is the software used in (self-driving) cars. The delivery of such software needs to be extremely secure and runs on a very unique architecture. For these, and embedded systems, DevOps faces the challenge of combining legacy systems with continuous delivery [13].

1.1 Research Questions

As seen, high levels of automation, collaboration, and correct tools for the job lead to shorter lead times. However, it remains unclear whether these technologies and practices influence the performance of organisations. Furthermore, some scholars raised the question: "Which practices are best for which kinds of systems in which kinds of organisations?" [1]. Therefore, the main research question is as follows:

What are the effects of DevOps practices and technologies on organisational performance?

Based on this question, the following guiding questions are identified:

- Which technologies are utilised for which DevOps practices?
- Which metrics are used in practice for measuring performance in the context of DevOps?
- What combination of DevOps practices and technologies are positively related with performance?

1.2 Contribution

Answering the research question helps in understanding how certain DevOps practices and technologies impact organisational performance. The contribution of this study is twofold. First, organisations, and more specifically DevOps practitioners at these organisations, are given recommendations on what practices to adopt to improve: 1) DevOps maturity, 2) software delivery performance, and 3) organisational performance. Second, the outcome of this research can be used to fill the current knowledge gap between DevOps practices and technologies, and organisational performance. The recommendations given in this research can be explored further in future research.

1.3 Document Structure

The remainder of this document is structured as follows. Chapter 2 discusses relevant scientific (and grey) literature to define the concept of DevOps together with its tools and practices. The effects and challenges of implementing DevOps within organisations are also described. Chapter 3 focusses on the methodology used within this research to answer the research question. It explains why a survey is used, the contents thereof, and the distribution strategy. In Chapter 4 the data gathered from the survey is described and depicted. Chapter 5 discusses and interprets the results and gives recommendations for DevOps practitioners. Last but not least, Chapter 6 finalises this research with a conclusion and pointers for future work.

Chapter 2

Related Work

Traditionally, the software development and IT operations departments worked independently of each other and communicated mostly via ticketing systems [5]. Knowledge concerning processes, tools, or practices was not shared, generating business silos. Although the development team focussed on delivering new software (perhaps using Agile), IT operations focussed on system stability, thus not letting through every change [5]. While this does ensure high quality code and a stable system, time-to-market for most changes is protracted. Adopting and leveraging the benefits of a new concept called DevOps, could combat these challenges.

This chapter reviews scientific literature to define the concept of DevOps together with its tools and practices. Next, the effects and challenges of implementing DevOps within organisations are discussed, as well as organisational routines.

2.1 What is DevOps?

To better understand the practices, tools, and technologies related to DevOps, first a clear definition of DevOps is required. However, the concept of DevOps has been difficult to define [3] and thus many definitions of DevOps exist [4, 5]. An unclear definition of DevOps can hinder its implementation [15]. Moreover, DevOps can be viewed through three different lenses: engineers, managers, and researchers [5]. Although some scholars have attempted to clarify DevOps concepts and practices [3], it still remains unclear how DevOps can be adopted effectively.

Most existing definitions mention that DevOps is either a paradigm, a method, or a set of practices that combines Development and Operations, bridges the gap between the two, and enables collaboration and communication. In their research, Jabbari et al. [4] define DevOps using the common components in other scholars' definitions. All definition found in scientific literature can be found in Appendix A.

Since this research focusses on the effects of DevOps on the performance of organisations, the definition by Balalaie et al. [22] is most fitting:

"DevOps is a set of practices which not only aims to decrease the time between applying a change to a system and the change being transferred to the production environment, but also insists on keeping the software quality in terms of both code and the delivery mechanism as one of the key elements in the development process."

Although this definition does not address collaboration and bridging the gap, it stresses the importance of a fast time-to-market and the ensurance of high quality products. As mentioned in the Chapter 1, organisations require efficient channels since their surroundings are ever-changing.

DevOps typically occurs on three levels: individual, team, and department [23]. Individuals often take a specific DevOps role: DevOps engineer or DevOps coach. Most organisation have "DevOps teams" containing members with different skill sets. Lastly, the borders between the Development and Operations departments are fading.

2.1.1 Key DevOps Components

DevOps consists of two core elements [5, 7, 15]: 1) collaboration between software development and IT operations, and 2) automation to manage deployments and the architectural environment. By easing communication between the two teams, software can be released in much shorter cycles [7, 24]. This, in turn, can lead to higher customer satisfaction and profitability [14].

Humble and Molesky [25] propose four key components of DevOps: Culture, Automation, Measurement, and Sharing, or CAMS for short. Together, these components align the incentives of all stakeholders: development, IT operations, and quality assurance. Scholars have later added the Lean component [26]. Together, these components are known as the CALMS framework, and is used to assess how well an organisation has adopted DevOps, or to measure success during DevOps transformations [27]. Figure 2.1 is a graphical representation of the framework. Each CALMS component will now be explained in detail.

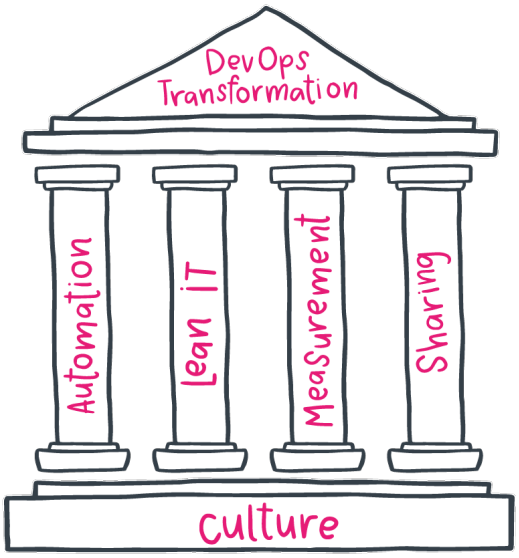


Figure 2.1: *CALMS framework for DevOps (from [28]).*

Culture

Although the practical side of DevOps involves practices and technologies, "DevOps is just another buzzword if you don't have the right culture" [29]. A key part of im-

plementing DevOps is therefore a shift in culture towards "collaboration between development, quality assurance, and operations" [13]. Business silos need to be broken down and a collaborative way of working should be instantiated [29].

One way of doing so is building a transparent culture and sharing knowledge [23, 25]. Developers and operators should empathise with each other to understand each others needs [30]. Members of the operations teams should be involved in the design of applications and be present during planning meetings to share their knowledge. Moreover, development teams should learn about production environments in case of critical incidents [25].

Humble and Molesky [25] suggest moving away from project-oriented teams and use product-oriented teams instead. This ideology is shared by Wiedemann et al. [14]: instead of delivering complete projects, teams focus on building features instead. This way, organisations can continuously deliver new software – and therefore value. Instantly switching to a product-based approached can be too much, however [27].

Another aspect of the DevOps culture is shared responsibility [3, 29]. Instead of "throwing the software over the fence", both development and operations are invested in the success of the product. This, in practice, results in better ways to deploy and maintain software. Putting development and operations staff in the same room will help increase collaboration. The aforementioned handovers are discouraged and team efforts reduce a culture of blame [29]. Instead, team members should be encouraged to experiment and learn from mistakes [3]. Shared responsibility and co-locating thus improve communication and collaboration between teams.

With new teams being formed, new responsibilities arise and new tools and practices are introduced. For these teams to be successful and work effectively, teams should be autonomous [29]. Moreover, teams should be allowed to choose tools based on their own skills and knowledge [23]. This way, teams can make better decisions without having to wait for approval. This does, however, require sufficient trust from higher level management. Risks should also be managed. Open channels of communication within the entire organisation [27] and new practices such as infrastructure as code help facilitate the changes by providing transparency.

Automation

The abovementioned shift in culture is followed by a change in business processes. Automation, by definition, removes manual work and creates repeatable business processes or functions. By removing "manual error-prone repetitive tasks" [3], errors are reduced and employees can be more creative and productive.

Humble and Molesky [25] argue that this change is possible with the right tools, enabling the practice of infrastructure as code. Automating processes will then result in shorter lead times and an increase in usable feedback [24]. This fast and frequent delivery of software updates is called continuous deployment [31]. As processes are usually followed by other processes, this too should be made possible under DevOps. Within DevOps, a pipeline models the process of building and releasing software, starting at development and ending with deployment. Pipelines are often run on cloud-based infrastructure and use various open-source tools. I will explain more about pipelines in Section 2.3.

Every change made by developers finds its way through the pipeline. Automated

tests will check if the modified code contains bugs or introduces conflicts. If all tests return positive, a new release candidate is created and is oftentimes deployed on a staging environment. The production environment is only one mouse click away. The pipeline thus gatekeeps the production environment [8] by allowing only valid code to pass through.

A benefit of such automation is *configuration as code*. Teams can create small, independent pieces of software – often called microservices – making them more reliable and easier to update and maintain [27]. Since these microservices have little dependencies, they can be deployed almost everywhere. Not only application code can be stored under version control [21], the same yields for infrastructure or other configuration settings. A useful side-effect of automation is that it results in up-to-date documentation [29]. Scripts used in automated testing or deployment can be used by both developers and operators to change the behaviour of infrastructure, for example. When everything is stored as code and under version control, older versions of applications and infrastructure can be instantly created in case of emergency [25].

Automation also helps increase collaboration [29]. As tasks (e.g. testing and deployment) are automated, human errors will decrease and free up time for other tasks automation cannot perform.

Lean

As mentioned in Chapter 1, DevOps practices find their origin in Lean, Agile, or The Toyota Way. The amount of Work in Progress is decreased by reducing batch sizes (software features). Software products are broken down into different functionalities and shipped one at a time. Work is only done when it is included in the current sprint [8]. Automation, as seen above, reduces manual work and increases quality by testing continuously.

Another aspect of the DevOps mindset is to improve wherever possible [27]; sprint retrospectives improve team performance and A/B testing improves products. By creating a minimal viable product (MVP) and improving continuously, users will be able to provide valuable feedback.

Continuous improvement also means dealing with failures or errors. Logging and monitoring tools provide real-time information about systems and infrastructure. Instead of pointing fingers, teams re-asses current processes and emerge stronger.

Measurement

In DevOps, measurement is defined as "monitoring high-level business metrics such as revenue or end-to-end transactions per unit time" [25]. For day to day activities this comes down to choosing the right metrics to measure. Following Agile principles, these could include: lead time, recovery time after failure, or number of customers [21, 27]. These business-focused metrics and their visualisations keep team members and others stakeholders informed at any given moment. With the right information, better decisions can be made. Metrics can also be used to identify bottlenecks, and thus keeping the process Lean [8]. Measurable data (e.g. conversion rate) is thus the determinator of success [30].

Aside from using the data to make informed decisions, it should be shared with other teams and departments [27]. Sharing information and open communication ensures all departments are on the same line, further breaking down business silos.

All data and metrics collected while operating the product or service can be seen as feedback [3]. Aside from the abovementioned metrics, this feedback can include how and when customers use services. By continuously monitoring customer behaviour, feedback loops for planning and development are created [30]. Developers gain real-time information about the software they build, making it easier to optimise [3]. Moreover, a continuous stream of feedback allows for true experimentation [3, 15].

Application monitoring is another way to look at measurement. Continuously monitoring production systems is used to ensure stability. When implemented correctly, quality assurance is increased [7]. An important aspect of DevOps is to build the required production environments alongside the actual application. As seen, microservices allow teams to develop smaller pieces of software. Managing these requires a pipeline that automatically triggers various monitoring tools [8].

Sharing

As mentioned in the Culture component, sharing knowledge is one of the main ways to break down business silos. The reorganisation of project teams into product teams delivering features instead of projects improves collaboration [8]. Developers, alongside operators, are now also in control of deployment and maintenance. With shared responsibilities and pains in case of crisis, successful releases brings teams closer together [21, 25]. Moreover, co-location and non-electronic communication increase the social aspect [8].

Like most aspects of DevOps, sharing (knowledge) is aided by the introduction of new tools. Forsgren et al. and other scholar [15, 32, 33] are advocators for the practice of 'version control for everything'. Alongside increased transparency, tools and technologies are easily shared across different teams. Developers can thus experiment with different tools and share their results with the rest of the organisation [8]. The aforementioned DevOps pipeline can also be used as a shared workflow when visible to everybody. Other tools that foster collaboration are chat and wikis [15, 34].

Lastly, infrastructure as code not only benefits automation but also improves sharing development environments. Development teams can easily share their setup and implement the same practices [8]. Virtualisation tools (e.g. Docker or Vagrant) can be used, for example, to quickly share environments without the issue of package and infrastructure requirements [9, 35]. Choosing an IaaS provider, however, comes with the difficult task of evaluating costs and performance [36].

2.2 DevOps Models in Scientific Literature

Leite and colleagues [5] created a conceptual DevOps framework containing fundamental concepts. The concepts are grouped into four categories: process (business related), people (skills and collaboration), (continuous) delivery, and runtime (stability and reliability). Using the aforementioned lenses (engineers, managers, and researchers [5]), the process and people categories would fall under management, and

the remaining two under engineers. However, the article distinguishes between developers and operators within the engineer group. For each category, a conceptual map was created with the most important concepts related to the category. From these maps, the most important concepts are: frequent release process with a short feedback cycle, cross functional teams with a culture of collaboration, tools and automation, and continuous monitoring.

A similar study was done by Bou Ghantous and Gill [15], showing matching results. One difference is the notion of an automated pipeline within the major concepts. The research identified twenty DevOps practices and twelve categories of DevOps tools. The most commonly found practices are: automating the entire development pipeline, automated test reports, and sharing of monitoring logs. Most practices can be grouped into the following categories: automated development pipeline (including testing), knowledge sharing, and version control for everything.

An exploratory study by Lwakatare et al. [30] used interviews to identify DevOps practices from practitioners. The common dimensions (collaboration, automation, and monitoring) came forward in their research as well. However, culture and collaboration are separated and a measurement dimension is added. Here, collaboration is seen as re-dividing the roles within teams, as responsibility increases. Moreover, collaboration requires more cooperation and involvement. Culture focusses on empathy, work environment, and respect. Useful metrics are used to incentivise both development and IT operation teams.

Both Forsgren et al. [21] and Ur Rahhman and Williams [37] stress the importance of security within DevOps. By "shifting left" on security [21], security and its practices are considered in the earliest phases. Security practices can be in the form of training or automated and non-automated tools [37]. For automated tools, monitoring and testing are used the most. This corresponds with the research described above, where both tools are included in the pipeline. Non-automated security practices include requirements analysis and configurations, as well as policies and manual tests.

2.2.1 DevOps Maturity Models

Although not all scholars [21] are keen on maturity models, they are a popular tool to assess the maturity of organisations. After reviewing both scientific and grey literature, several DevOps Maturity models have been identified [38, 39]. Zarour and colleagues [39] benchmarked seven maturity models based on their levels, dimensions, and application. Most models share the Collaboration, Automation, Monitoring dimensions and their levels are progressive. It is important to note that most models are either found or based on grey literature. Eficode's model [38] covers most aspects of the aforementioned CALMS models and can be filled out unambiguously. For these reasons, it will be used within this research. More details regarding the model can be found in Chapter 3.

2.3 DevOps Practices & Tools

Merriam-Webster defines practice as "to do or perform often, customarily, or habitually" [40]. Brunnert and colleagues [20] define DevOps practices as any technique

that enables the DevOps goal. Although many scholars have written about DevOps practices [1, 5, 13, 30], DevOps principles and practices are not well defined within scientific literature [13]. By reviewing relevant literature, 47 DevOps practices have been identified and combined where possible (see Appendix B). The most common and, for this research, relevant ones can be found in Table 2.1.

Table 2.1: *Most common DevOps practices.*

#	Practice	References
1	Automated and continuous deployment throughout entire pipeline	[4, 9, 11, 15, 33, 34, 41]
2	Make small and continuous releases	[32, 41, 42]
3	Developers get feedback based on releases	[32, 42]
4	Create development sandboxes for minimum code deployment	[15]
5	Everything is stored as code and under version control	[4, 9, 15, 32, 33]
6	Integrated configuration management	[4]
7	Automated and continuous testing in development and staging environments	[4, 15, 32, 33, 42]
8	Reduce the time it takes to test, validate and QA code	[34]
9	Code reviews are change based	[9]
10	Automated and continuous monitoring of applications and resources	[4, 9, 22, 32]
11	Automated dashboards that include health checks and performance	[4, 32]
12	Support configurable logging that can optionally be turned on/off as needed	[32]
13	Use trunk-based development over long-lived feature branches	[9, 33]
14	Use Test driven Development where all code has unit tests	[42]

Tools and technologies are a requirement when automating DevOps processes [5, 13]. Tools within the context of DevOps have been defined as tools used for collaboration, continuous delivery, and reliability [5]. According to Leite et al. [5], DevOps tools can be defined as: "tools pursuing one of the following goals: 1) assisting cross-departmental collaboration, 2) enabling continuous delivery, or 3) maintaining software reliability". For each phase within DevOps, a plural of tools are available [43]. The vast majority of tools can be explained by the custom user experiences practitioners demand [43]. Suppliers of development tools feel pressured to deliver more specialised tools. Thus, many technologies claiming to help relieve the stresses around development and deployment arose. Their common focus lies on speeding up the delivery process [13]. After scanning relevant literature, DevOps tools can be grouped into 25 categories. The categories and their references can be found in Table 2.2. All identified tools can be found in Appendix C.

Although tools play an important role within (automating) DevOps, they should not function as the key to enable it. When introducing DevOps into organisations, team structure is more important than tools [5]. With the existence of many DevOps tools, careful considerations should be made on deciding which one to adopt, as integration can be difficult [15]. Moreover, developers should be allowed to make their own choices in which tools they use [21].

The objective of the following sections is to establish relationships between the concepts found in the aforementioned scientific literature and tools used by DevOps practitioners.

2.3.1 Collaboration

As seen in the Culture aspect of the CALMS model, DevOps requires a culture shift involving collaboration across different business silos. Different departments within organisations thus need to share processes, practices, and knowledge [5]. One practice involved in sharing knowledge is to provide a continuous collaboration system

Table 2.2: *Functionalities of DevOps tools.*

#	Functionality	References
1	Build	[13, 15, 34, 35, 44]
2	Cloud Computing	[9, 11, 15, 34, 35, 45]
3	Code Analyser	[9, 34, 35, 46]
4	Collaboration	[15, 34]
5	Configuration Management	[6, 9, 13, 34, 35, 44, 47, 48]
6	Continuous Delivery	[6, 11, 13, 15, 35, 47]
7	Continuous Integration	[1, 6, 9, 11, 13, 15, 35, 44, 46, 47, 49]
8	Database Management	[6, 15, 34]
9	Delivery Server	[49]
10	Deployment	[11, 34, 35, 47, 48]
11	Documentation	[12]
12	Hosting	[11]
13	Infrastructure as Code	[11]
14	Knowledge Sharing	[34]
15	Logging	[9, 13, 15, 34, 35, 44]
16	Monitoring	[9, 13, 15, 34, 35, 44]
17	Process Analysis	[9]
18	Project Management	[12, 35, 46, 49]
19	Scripting	[34]
20	Security	[34]
21	Software Package Manager	[6]
22	System Performance	[50]
23	Testing	[9, 11, 15, 34, 35]
24	Version Control Management	[6, 9, 11, 15, 34, 35, 46, 49]
25	Virtualisation	[6, 9, 34, 35, 45, 51]

in real-time [15]. `GitHub`, `GitLab`, `Wikis`, or `Rocket Chat` are examples of tools used to share knowledge. With `Wikis` anyone can update or create topics, which are easily searchable. Over time this will create a large number of documents – often referred to as knowledge base – with up-to-date information.

Open communication channels between members from different teams or departments aids in collaboration [8]. `Slack` is a communication tool containing different channels for tools, files, and people. Different channels can be created for different use cases or projects. It also includes one-to-one messaging, a replacement for traditional e-mails.

Other practices improving collaboration involve the empowerment of employees, especially developers [30]. As developers gain new responsibilities, their knowledge and skill set should be extended. Organisations need to choose the right tools and methods for empowerment. Two practices are closely related to this aspect [30]: 1) increasing the scope of responsibilities, and 2) intensifying cooperation and involvement. By increasing responsibilities and working closely together, developers become pro-active in learning and sharing new skills.

Infrastructure as code functions both as documentation and a way to improve transparency. By using version control management software (e.g. `Git`) and host the code online (e.g. `GitHub`, `BitBucket`, or `GitLab`), everybody has up-to-date documenta-

tion. Using commits and pull requests developers can update code and documentation alike.

Another tool to promote collaboration is source code management [5]. Source code management is at the core of continuous integration and delivery, as well as automation. Some scholars are in favour of storing everything as code and under version control [13, 21]. By not only storing code, but also artifacts [5], developers gain insight into how their code runs in production environments. `Git` and its online platforms (e.g. `GitHub`) are commonly used for this task. Online platforms have the benefit of providing visualisations of changes and often include issuing systems [5].

2.3.2 Automation

The combination of software development and IT operations results in a set of practices, methods, and technologies: a toolchain. DevOps practices (e.g. container management, continuous integration, testing, and monitoring) heavily rely on tools and technologies [1, 15]: numerous tools exist for each practice. The actual route a piece of software undertakes from coding to production is called a pipeline.

A pipeline – another manufacturing concept used in IT – consists of tools from the toolchain which are connected and form a one-way path. Software follows the pipeline and moves through different environments (e.g. testing and production). Developers can continuously update software, and in a more efficient manner compared to existing methods [34]. Some scholars have named this pipeline the "DevOps Platform" [52]. Common categories for tools and practices within a pipeline include: programming, building, testing, and releasing. A well-created pipeline thus allows for short lead times.

To ship new software with high quality on a regular basis, most steps within this process are automated [13]. Automation within the DevOps infrastructure heavily relies on the correct implementation of tools and the right configurations for all software-related items. Effective automation begins by using version control for everything: code, tests, and infrastructure [21, 30]. In addition, automated runtime monitoring is used to improve performance, availability, and security [53]. These processes encourage software developers to write better code [1]. In addition, changes can be rolled back, requiring a specific software and information architecture.

Within the concept of automation, three 'continuous' practices are common: integration, delivery, and deployment. Stahl and colleagues [41] explain them in an understandable manner. Continuous integration is the practice of frequently (e.g. daily) integrating the changes of multiple developers. Continuous delivery is the practice of treating every change as a *potential* release candidate evaluated by the pipeline that *could* be deployed. Note the emphasis on potential and could: for business reasons, deployment can be held back. Continuous deployment, then, is the practice of frequently deploying release candidates to production.

As mentioned, the practice of using infrastructure as code is central to automation. Applying this practices requires developers to establish the infrastructure in a reproducible and programmatic manner, and stored under version control [30]. Configurations and other practices from software development can be applied here as well.

2.3.3 Continuous Delivery & Deployment

Continuous deployment and delivery are the most important practices within DevOps [4]. Using these practices, developers can focus solely on their project and do not require input from other teams [1]. A common practice found in literature is enabling automated and continuous deployment throughout the entire pipeline [4, 9, 11, 15, 33, 34, 41]. Continuous delivery can be viewed from two angles: engineering and business [23]. Engineers construct and automate the pipeline with the right tools, whereas the business determines which departments are involved.

As seen, source code management is at the core of Continuous Delivery. Other tools that are required to enable automated software delivery are so-called build tools. Within the build phase, Continuous Integration (CI) tools are used to integrate software from different developers [5]. CI tools can be configured to test the system every time a developer commits a change within the versioning system. Possible errors can thus be found before the software moves to the next step within the pipeline. `Jenkins` (open-source) and `Bamboo` (Atlassian) are examples of popular CI tools.

Each programming language has its own build tools and test frameworks [5]. Build tools (e.g. `Maven` or `Gradle`) perform all tasks to build a shippable product, an artifact. It fetches the required software dependencies, checks for broken code, runs various tests, and can perform custom tasks (e.g. send a success message via Slack), improving software quality [13]. When an artifact is created, product owners can put these into production with a single button click.

An important step in this pipeline is testing. Unsurprisingly, many scholars [4, 15, 32, 33, 42] favour the practice of 'automated and continuous testing in development and staging environments'. Tools for testing are test frameworks written in the same language as the source code of the application. Test frameworks can perform unit tests or even automate human behaviour [5]. When tests fail, the software will not continue through the pipeline and no artifact will be created. Some example of test frameworks are: `Cucumber`, `Selenium`, and `Junit`. Such frameworks provide developers with feedback regarding the correctness of the software. Feedback can also come from code analysers (e.g. `JS hint`, `Jacoco`, or `FindBugs`). These tools scan code during development and compare it to certain rules that are agreed upon beforehand.

Continuously delivering software affects the required architecture of the overall product. One architectural style often used when needing to deploy software continuously, are microservices [1]. Microservices are small, independent, and autonomous pieces of software that focus on a single process [54]. Typically, a microservice is built and maintained by a single group of developers, carrying all responsibility. The full-stack development method fits best in this situation [13], since every developer requires the know-how of the entire microservice. A requirement for microservices is that developers understand they contribute to the entire system. Wherever possible, microservices (and DevOps) should be cloud based [13]. A major benefit of microservices is that they are independent of each other. Updates can be carried out continuously, without requiring changes in other systems. Other benefits include: 1) using different technologies for each microservice, 2) microservices are resilient, 3) microservices scale well, and 4) microservices can easily be replaced.

However, scholars remain unclear whether microservices should be versioned [5]. In addition, the dilemma between containerisation and virtualisation exists [5, 13].

With these technologies, production environments can be created for use in development. The focus of virtualisation lies on security, but lacks platform-compatibility. Containerisation (e.g. using `Docker`) can run on any platform and can be version controlled. Another tool used for virtualisation is `Vagrant`. Benefits of containerisation include [45]: 1) services run on the same environment in both development and production, 2) all dependencies are included in the configuration of the container, and 3) updating one service has zero impact on other services as they do not interfere. Moreover, containers simplify the architecture and are more resource efficient [55].

The key practice during the deployment stage is to use configuration for everything [13, 21]. This allows environments and architectures to be shared and tested using version control practices. Minimising the differences between development and production environments also reduces the chances of bugs and other problems [13]. To enjoy the benefits of automated and continuous delivery, systems need to be connected [13]. This required architecture can be achieved using the aforementioned microservices. Tools for Continuous Delivery are: `Puppet`, `Chef`, or `Ansible`.

2.3.4 Measurement

As discussed, within DevOps there are two main ways to look at measurement: 1) measuring high-level business metrics (or KPI's) keeps team members informed and allows for better decision-making, and 2) real-time monitoring of production systems increases stability.

Monitoring systems track aspects such as: network traffic, temperatures, memory, and other non-functional properties [5]. Other functionalities that come with monitoring tools are: self-healing, alerting when problems are detected, and log management. Akshaya et al. [44] distinguish two categories: system and network tools.

Two practices involving monitoring have been identified from literature: 1) applications and resources are automatically and continuously monitored [4, 9, 22, 32], and 2) automated dashboards including health checks and performance metrics [4, 32]. Tools commonly used for monitoring are `Zabbix`, `Nagios`, or `Cacti` [13]. These tools can either be offered as a Service or through web interfaces.

Logging is the practice of storing pre-defined data over a period of time. Log data can be used for different reasons, depending on the stage within the DevOps lifecycle [44]. For example, logger frameworks are used to debug in production, since logging is an important way to manage applications [13]. It can also be used for debugging during development, performance testing, and web analytics [44].

Hamilton [32] identified the practice of supporting configurable logging that can optionally be turned on/off as needed to debug issues. This way, certain information is logged to a designated log-file only when it is required. This is useful during the introduction of new features or when bugs arise. Some of the logging tools available include: `Graylog`, `LogStash`, or `Loggly`. Most cloud services include monitoring and logging functionalities by default [5].

2.4 Benefits After Implementation

Table 2.3 presents a short overview of benefits after implementing DevOps, according to literature.

Table 2.3: *Experienced benefits after DevOps adoption.*

#	Benefit	References
1	Automation of processes	[14, 15]
2	Shorter cycle times	[6–9]
3	More frequent releases	[10, 11]
4	Continuous experimentation and improvement	[3, 7, 8, 15]
5	Increase in stability	[8, 24]
6	Increase in quality	[6, 7, 9]
7	Improved collaboration and communication	[7, 12]
8	Better and happier employees	[3, 11]

DevOps enhances automation throughout the software delivery process [14, 15]. Automation not only reduces errors, but also improves the development process. This, in turn, leads to developers being more creative and productive [3].

Continuously delivering new software improves the time it takes services to reach users. Callanan and Spillance [6] found a 86% decrease in release cycle time – the time from 'development and testing complete' to 'released' – from weeks to hours, or even minutes. Moreover, the frequency with which new (smaller) releases are made, also increased [10, 11]. This positive effect on the cycle time was also found by other scholars [7–9]. For some organisations, the time-to-market improved with 20% after the introduction of DevOps [10].

Shorter release cycles also benefits customers: with less new features to focus on each release, users can provide better feedback [14]. Organisations could thus test with real customers and use their feedback, enabling continuous experimentation [3, 7, 15] and improvement [8].

According to some scholars, DevOps requires a "shift to product-based management" [14]. This translates into shifting from delivering complete projects to a continuous delivery of features. Riungu et al. [7] found that this leads to more features being implemented and an improvement of quality assurance.

Adhering to the practice of continuous integration, organisations can make small changes to software in production. This has the benefit of easier identifying defects and hotfixes can be deployed faster [6]. The overall quality of the product or service thus increases [9].

Another benefit of DevOps is an increase in stability [8]. Stability is achieved by automating tests and a culture of continuous improvement. Forsgren et al. [24] measure stability using the Mean Time to Recover (MTTR) metric. MTTR measures the time it takes to recover from a service incident [21]. As mentioned before, using version control and automated deployments roll-backs are simpler. High performers can restore service in less than one hour, whereas low performers may need up to one month [24]. Th MTTR of 'elites' is 2,604 times higher than low performers.

As DevOps requires breaking down the silos between the development and operations teams, better collaboration and cross-department communication are common

benefits of adopting DevOps [7, 12].

Callanan and Spillance [6] found that shifting to a DevOps mindset, required no extra costs (e.g. licensing, hardware or man-hours). Other scholars even found that after implementing DevOps, costs for development and operations decreased in 46% of the cases [10].

Lastly, DevOps can also improve employees' way of working, in turn improving their well-being. Teams become happier and are more engaged with the overall product [11]. By better understanding the needs and expectations, pointing fingers between developers and operators reduces.

2.5 Challenges Implementing DevOps

Table 2.4 shows a short overview of challenges that can potentially hinder the adoption of DevOps.

Table 2.4: *Experienced challenges implementing DevOps.*

#	Challenge	References
1	DevOps remains a vague concept	[3, 7, 8]
2	Shift in organisational culture required	[3, 7, 13–15]
3	Lack of communication	[7, 8]
4	Management approval required	[8, 14]
5	Employees gain new responsibilities	[3, 8, 15]
6	DevOps is very context dependent	[7, 13, 14]

Depending through which lens (engineer, manager, or researcher) one looks at DevOps, different challenges can arise [5]. Engineers want to continuously deliver new software, which requires new architectural structures. Managers face cultural challenges when trying to introduce DevOps into the organisation. Lastly, researchers try to understand the latest practices. Successfully implementing DevOps into an organisation is thus not without challenges.

Ebert et al. [13] identified four major challenges when adopting DevOps: 1) complex architecture needs to be broken down into small individual chunks (e.g. microservices), 2) configuration and build environments should be maintained and be visible for all, 3) legacy application life-cycles are converted into development environments, and 4) breaking down the silos between software development and operation teams.

While Riungu-Kalliosaari et al. [7] found numerous benefits of adopting DevOps, some implementation challenges were raised. When communication between teams is lacking (e.g. only electronically), not all knowledge and information is shared, possibly resulting in problems. The practices of continuous monitoring, for example, requires input from all teams involved. Development teams need to know how their software performs in production environments.

Implementing DevOps requires a significant shift in organisational culture [3, 7, 13–15]. This cultural shift can be more difficult than the technical implementation [14]. Teams and individuals gain new responsibilities and tasks [7]. Operations teams are suddenly connected to many other (business) functions. Hamunen [8] identified three

types of issues on a team level: 1) lack of trust, 2) lack of skills, and 3) lack of communication.

On a higher level, if management is not supportive of the changes, DevOps initiatives will likely never see the light of day. Lack of management support is therefore considered another challenge in DevOps [8]. Close collaboration between teams and strong leadership is therefore required. As culture is unique for every organisation, an approach that works for some, might not work for others. Integrating a culture of collaboration within team settings is a must [14].

Not all aspects and practices of DevOps are suitable in every situation [7, 13]. The context of which an organisation operates in, is unique. Therefore, a custom interpretation is required for each organisation [14]. Systems used within banks or healthcare organisations, for example, cannot use continuous delivery, as certain systems cannot be shut down easily. Changing legacy systems and older infrastructure to microservices is also difficult [13, 15]. Moreover, complex production systems are hard to clone to development environments. Automated testing is then less reliable.

In addition, some developers have a hard time embracing the practices of automating tests [8]. Originally, testing was not part of the developer role, so inexperience can lead to frustration. Moreover, developers need to learn additional skills to keep up with operations [3]. As writing tests does not add value, like building features, some even consider it waste. Actions to combat these challenges can include: making test templates developers can view and use, or requiring a certain level of code coverage before a change can make it to production. As automation requires new tools and practices, picking the right tools [8] and maintaining them [15] can be a challenge on its own. Quality and maturity levels of tools differ immensely and can become outdated quickly.

Some scholars have stated [3, 8] that DevOps, with all its definitions remains vague. In contrast, practitioners advocate that vagueness allows organisations to adopt a definition that works for them [14]. Misconceptions about DevOps include: DevOps is something that can be purchased, or DevOps is either a “guy” or a “team” [8]. There is (yet) no standardised package of practices suitable in every situation and many different tools exist. The concept of DevOps thus remains “unclear but also evolving” [7]. However, according to Leite et al. [5], team structure should be prioritised over tools. Similarly, Bass and colleagues [56] recommend alignment between DevOps and organisational goals.

2.6 Organisational Routines

Since practices, and more specifically DevOps practices, are hard to define, a more solid and scientific basis is required. Therefore, this section will discuss relevant literature regarding organisational routines and capabilities.

Becker [57] systematically reviewed the concept of routines. He identified the following characteristics routines have. First, routines are patterns, focussing on the concept’s consistency. By distilling concepts from relevant literature, Becker found that patterns consist of four different terms: action, activity, behaviour, and interaction. However, in economics and business, behaviour and interaction are considered subsets of activity. He therefore proposes two other kinds of patterns: activity patterns

and cognitive regularities. Recurrence, by definition, is the second aspect of routines. Activities need to be repeated in order for it to become a routine. The third characteristic is the collective nature: routines involve multiple actors and can be distributed across different locations (e.g. departments or physical locations). Dosi, Nelson, and Winter [58] consider routines to be on the organisational level. Since routines occur in a business setting, they are dependent on its context. The next characteristic is therefore context-dependence and specificity. Routines can be specific in three ways: historical, local, or relational. The historical aspect of routines means that they can change over time in a path-dependent way [57]. New information or feedback can restructure a routine. Lastly, routines start by one of two triggers: actor-related triggers or external effects.

Becker [57] also identified multiple benefits of routines on organisations. Routines reduce uncertainty by providing a rule-based way of choosing between different possibilities. Routines also provide stability in employee behaviour: when a certain routine provides sufficient results, employees will not seek different ways to do things. Moreover, changing a routine involves 'costs': the identification of new actions takes time and effort. Lastly, routines are a source of knowledge since they provide specific ways of carrying out activities. Such activities may include problem solving. Compared to other sources of knowledge (e.g. Wiki's or documentation), routines can store tacit knowledge [59]

Organisational routines can be seen as one of the building blocks of organisational capabilities [58]. Definitions of organisational capabilities fall in three categories [60]: 1) the ability to perform basic functional activities, 2) dynamic improvement to activities, or 3) strategic insights to develop novel strategies before competitors. Collis' [60] definition of capabilities is as follows: "the socially complex routines that determine the efficiency with which firms physically transform inputs into outputs".

Organisational capabilities have been found to be a source of competitive advantage [60, 61]. Salvato [62] found that a number of routines carried out by individuals within the organisation "allow organisations to successfully renew their core capability" [62]. He argues that organisations should learn to recognise valuable experiments to maintain their competitiveness. In a DevOps setting, experimentation could come from continuously interacting with customers through continuous integration and using their feedback.

Chapter 3

Methodology

As seen in the previous chapter, exploratory research already found a number of DevOps best practices and technologies. Different research studies found numerous benefits after adopting the DevOps approach [3, 6, 10, 11]. Moreover, a positive connection between DevOps practices and technologies and short lead times was also found [33].

To answer the research question – *What are the effects of DevOps practices and technologies on organisational performance?* – further "deepening" research is required. Confirmatory research is a paradigm used in science to gather objective information that can be generalised [63]. It requires detailed planning before data collection can begin. Confirmatory research is quantitative as it allows for generalisation. To be able to generalise for all of the population, input from a large number of DevOps practitioners is required.

3.1 Population & Sample

The first step in defining the population for this research is to determine the unit of analysis. The unit of analysis is the entity studied within a research project. As this thesis analyses the effects of DevOps tools and practices on organisational performance, the unit of analysis can therefore be defined as: organisations who are building software according to the DevOps methodology.

Not all employees are equally suitable to participate in this research. Hence, the population needs to be more precise. Only employees involved in the development, testing, or maintenance of software are relevant for this study. Such employees typically have one of the following roles: Software Engineer, Tester, DevOps Engineer, Security Engineer, or Automation Engineer. Managers or higher level executives are also included since they share different views on this topic. The same yields for consultants such as Agile coaches or external consultants.

The population can therefore be defined as: people working at software development organisations which follow the DevOps methodology, with a role similar to the ones described above.

To be able to generalise the results for the entire population, the aim is to gather a sample of at least 100 people who are characterised by the abovementioned aspects.

3.2 Survey

To reach a large number of people who fall within the population within the timeframe of this thesis, a survey is the most obvious choice. Aside from being easy to distribute, anonymous internet-based surveys reduce desirability biases [64]. Another benefit of using a survey is that data can be analysed quickly after collection [33]. Moreover, surveys are often used in this field of research, for example: the State of DevOps reports [24, 65–67], Humble et al. [33], and Forsgren [68].

The web-based survey will be made using the software provided by the university: Qualtrics.

3.2.1 Survey Design

The survey consists of the following six categories: 1) Demographics, 2) Transformation, 3) Maturity Models, 4), Tools & Practices, 5) Impact of DevOps, and 6) Organisational Performance. The full survey containing all questions can be found in Appendix E.

The survey closes with a thank you message and a link to a separate survey to enter one's e-mail address. Respondents can leave their e-mail address if they are interested in the outcomes of this research. Separating the e-mail address from the answers guarantees the respondents' anonymity.

Demographics The survey starts by collecting data about the participant to ensure a diverse enough sample of the population. This category includes questions regarding the industry, the role within and size of the organisation, and years of experience with DevOps. These questions also function as categorisation questions; there could potentially be differences in results between different industries or company sizes.

Transformation In this category the participants are first asked if their organisation has undergone or is currently undergoing an agile or DevOps transformation. If not, the survey skips to the next section, skipping the rest of the questions. Otherwise, this section continues and asks the respondents about the duration of the transformation and the frameworks used during the transformation.

Maturity Models While not all scholars agree on the usage of maturity models [24], they are a popular tool to measure the current state of an organisation. To measure the Agile and DevOps maturity of organisations, the third section contains questions regarding two maturity models. First, questions based on Laanti's Agile Maturity Model [69] ask the participants to rate their maturity on three organisational levels: portfolio, program, and team. The possible answers to these questions range from 'Beginner' via 'Fluent' to 'World-class'.

Eficode [38] developed a DevOps Maturity Model consisting of six enterprise areas: Organization & Culture, Environments & Release, Builds & Continuous Integration, Quality Assurance, Visibility & Reporting, and Technologies & Architecture. Each area can be rated according on a scale ranging from level 1 to level 4. All questions are accompanied with a table of the corresponding level/area.

Practices & Tools This section explores which tools, technologies, and practices are used in the day-to-day work of the participants. Questions include where software is hosted, how it is deployed, if open source software is used, and what their DevOps pipeline looks like. In addition, questions regarding capacity allocation measure what people spent most of their time on. Next, multiple survey blocks list the most common practices (from Table 2.1), grouped by category. The second practice is split up into two subpractices to better understand their potential impact. For each category, the respondents are asked how often they apply each practice. The answers are on a 7-point Likert scale. The last questions in each category – shown only when they apply at least one practice in the category – require input on the tools and technologies that are used.

Impact of DevOps Forsgren et al. [21] created a latent construct for *Software Delivery Performance*. This construct is made up of four questions regarding lead time, deployment frequency, mean time to restore (MTTR), and change fail percentage. These questions are included to compare with their research and have shown to have a positive relation with organisational performance.

Second, to evaluate the perceived impact of DevOps on the participants and their team members, they are asked to describe opinions on these metrics, as adopted from Lanti [69]: effectiveness of development, quality of the product, customer satisfaction, collaboration, work being more fun, work being less hectic, work being more organised, and earlier detection of bugs/errors. Additonal metrics regarding customer satisfaction are added, e.g.: usefulness of the product, usability of the product, and predictability of delivery. The questions ask to input the percentage of change the impact has, ranging from -100 (negative impact) to 100 (positive impact). The negative values are in place to counter bias.

Organisational Performance The last section focusses on measuring organisational performance. Commonly used questions for this metric are the ones adopted from Widener [70]. Participants are asked to rate their organisation's relative performance on multiple dimensions. This survey uses the following dimensions: overall performance, overall profitability, customer satisfaction, quality of products and services, operating efficiency, and achieving organisational goals. For each dimension, a question asks how well the organisation met its goals regarding that dimension. The answers are on a 7-point Likert scale, ranging from 'Performed well below goals' to 'Performed well above goals'.

3.2.2 Distribution

The survey was distributed through the personal network of the author and social media platforms such as Facebook and LinkedIn. By creating interesting posts on the platforms, people will share the posts with their network, thus gaining increased visibility. This way, posts will snowball and reach as much people as possible. Next, unique posts will be created in relevant DevOps groups on the social media platforms. A benefit of these groups is that all people are interested in the topic. In addition, people with relevant DevOps experience were targeted directly by messaging them via the

platforms.

3.2.3 Quality Assurance

As described above, the survey will be used to measure predefined metrics. Most of them are based on related work by other researchers – e.g. Forsgren et al. [21], Laanti [69], and Widener [70]. Questions based on their work are tested for both reliability and validity, meaning the questions measure exactly what they should. This way, the quality of the survey is ensured.

3.3 Data Analysis

The first step after the data collection is to export the data. Qualtrics allows for exporting the data to different files, including a `csv` file.

Before the data can be analysed, it first needs to be sanitized. The data sanitation process consists of the following steps. First, all partially filled in responses will be deleted from the data. Then, all irrelevant columns (e.g. start time, progress, or response id) will be removed to simplify the analysis. Next, all text inputs will be scanned for spelling mistakes to avoid duplicates of the same answers. Lastly, long sentences will be condensed and labelled for easier access.

The analysis will be done with Python using the `pandas`, `scipy`, and `numpy` packages. Visualisations will be made using the `plotly` package. The clustering is done with the K-means algorithm from the `sklearn` package. K-means was used with different values for K, until distinct clusters were found. Correlations were made using the Pearson correlation, a commonly used statistic. The correlation and p-value were calculated using the `scipy` package. After the sanitized `csv` file is imported, visualisations were made and the clustering and correlation methods were executed.

3.4 Reflective Workshops

After the collection period had ended and the initial results were visualised, some of these results were used in three interactive and reflective workshops. Two of them were held during the 2021 RTE Summit, the last one took place online and was organised by IPMA Connect.

Participants of the workshops were grouped around the different results and were asked to reflect on and discuss the outcomes. Using (digital) sticky notes, participants described their observations, came up with reasons for those observations, and related it to their experience. They then prepared a short plenary to share with the entire group. After all groups had shared their discussions, a more general discussion about the results, impacts, and experiences took place. Some of these discussions are included in Chapter 5.

Chapter 4

Results

The survey was active in the period from June 30 to November 5, spanning 128 days. Within this time, 337 people started the survey and 123 completed it. The response rate of the survey is therefore 36.5%. This chapter describes and depicts the data gathered from the survey.

4.1 Who Are the Respondents?

To better understand the respondents, first the demographic information is displayed. Next, to compare the results of this survey with comparable previous studies, questions regarding a DevOps or Agile transformation were included in the survey.

Table 4.1 displays the most common roles of the respondents. The three most common roles are Software Developer/Engineer (30.1%), IT Operations/Infrastructure Engineer (22.0%), and Automation Engineer/Expert (8.9%). The roles categorised under Other only had one or two occurrences and are therefore not displayed.

Table 4.1: *Most common roles of respondents.*

Role	Respondents	Percentage
Software Developer/Engineer	37	30.1%
IT Operations/Infrastructure Engineer	27	22.0%
Automation Engineer/Expert	11	8.9%
Product Manager	7	5.7%
Project Manager	5	4.1%
External consultant	4	3.3%
DevOps Engineer	4	3.3%
DevOps Coach	4	3.3%
Sponsor	3	2.4%
Data Engineer	3	2.4%
Other	18	14.6%
Total	123	100%

The primary industry of the respondents can be found in Table 4.2. The Technology (32.5%) sector is represented the most, followed by Financial Services (15.5%). Retail,

Consumer, & E-commerce (8.1%) ranks third. The Education (1.6%) sector is represented the least.

Table 4.2: *Primary industry of respondents.*

Industry	Respondents	Percentage
Technology	40	32.5%
Financial Services	19	15.5%
Retail, Consumer & E-commerce	10	8.1%
Other	9	7.3%
Insurance	8	6.5%
Telecommunications	8	6.5%
Energy & Resources	7	5.7%
Government	7	5.7%
Industrial & Manufacturing	6	4.9%
Media & Entertainment	4	3.3%
Healthcare, Pharma & Life sciences	3	2.4%
Education	2	1.6%
Total	123	100%

Figures 4.1 and 4.2 show the size of organisations (number of employees) and the years of DevOps experience, respectively. The largest groups of respondents work at organisations with 20-99 employees (24.0%), 100-499 employees (20.7%) or 10,000+ employees (20.7%). Only a small percentage of the respondents work in organisations with less than 20 employees: 8.2%. For DevOps experience, most participants have 3-5 years of experience (38.2%), followed by 1-2 years (22.0%), and 6-10 years (17.9%). 5.7% of the respondents have no DevOps experience.

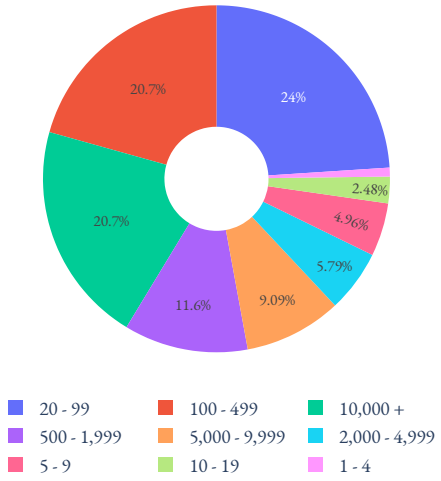


Figure 4.1: *Organisation size.*

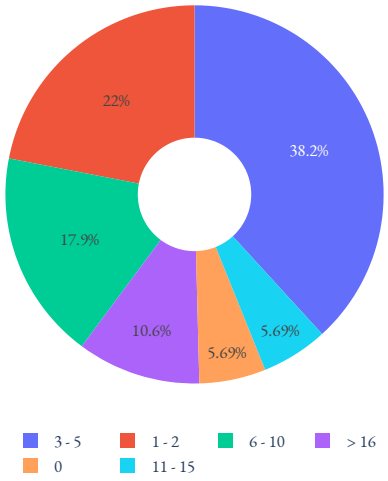


Figure 4.2: *DevOps experience.*

As shown in Table 4.3, 85.4% of the respondents indicated that their organisation has undergone or is currently undergoing a DevOps or Agile transformation. A small portion of the respondents (4.9%) said that their organisation is about to start a transformation. The remaining 9.8% will thus not be taken into account for remainder of this section.

Table 4.3: *Currently ongoing DevOps or Agile transformations.*

Answer	Respondents	Percentage
Yes, completed a transformation	57	46.3%
Yes, currently undergoing a transformation	48	39.2%
No, but about to start a transformation	6	4.9%
No, not at all	12	9.8%
Total	123	100%

The (estimated) duration of the transformation is displayed in Figure 4.3. The largest two groups of respondents mention a duration between one and two years (29.7%) and between six months and one year (28.8%). The remaining respondents classified the duration to be between two and five years (17.1%) or less than six months (15.3%). 9.0% of the respondents estimated the duration of the transformation to be more than five years.

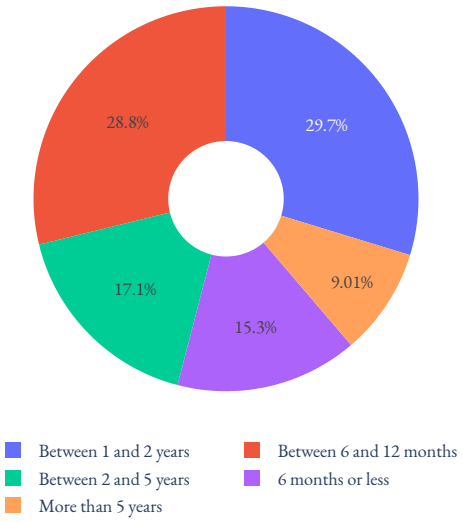


Figure 4.3: *Duration of the transformation.*

Figures 4.4 and 4.5 depict the Agile and DevOps frameworks used during the transformation, respectively. The most common Agile frameworks are the Scaled Agile Framework (26.1%), Enterprise Scrum (22.5%), Scrum of Scrums (16.2%), and Agile Portfolio Management (14.4%). 11.7% of the respondents used internally created methods, while nine percent of the respondents indicated they did not use a Agile framework during the transformation.

More than half (60.0%) of the participants also did not use a DevOps framework during the transformation. The most popular DevOps frameworks are CALMS (20.9%) and SAFe's CALMR (11.8%). Since the Scaled Agile Framework also includes DevOps topics, respondents might not need a separate DevOps framework. In fact, 37.9% of the respondents who used SAFe, did not use a DevOps framework.

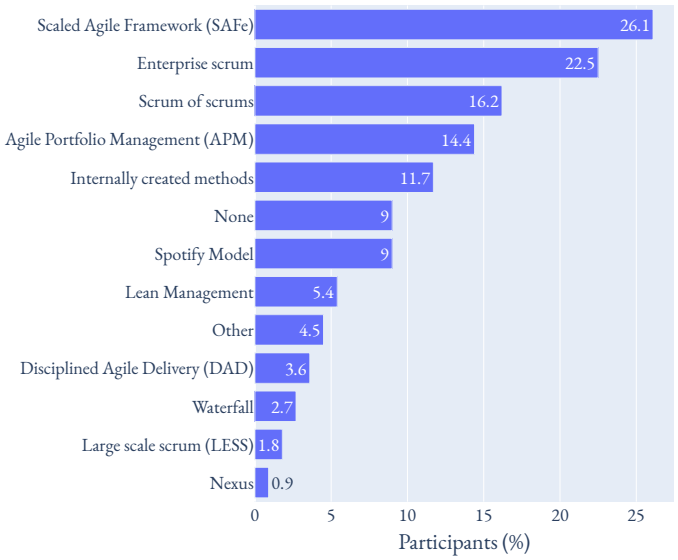


Figure 4.4: *Agile framework used during transformation.*

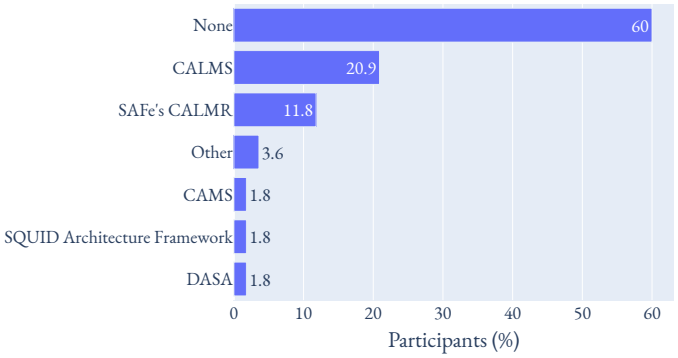


Figure 4.5: *DevOps framework used during transformation.*

4.2 DevOps & Agile Maturity

To be able to recommend certain tools and practices to DevOps practitioners, first a good understanding of the organisation's maturity is required. This allows us to link practices to maturity and make recommendations on improving maturity. DevOps and Agile maturity per industry can be found in Appendix D.

Figure 4.6 depicts DevOps maturity across six aspects as defined by Eficode [38]. In general, most respondents estimated their organisation to be on either Level 2 or Level 3. On the Organization & Culture aspect, Levels 3 and 4 were most common with 40% and 29%. The rest of the participants estimated their organisation to be on Level 2 (16%) or Level 1 (15%). On the Environments & Release aspect, Level 3 was most common (41%), followed by Level 2 (30%). The remaining respondents rated their organisation Level 4 (18%) or Level 1 (11%). 41% of the respondents rated their organisation Level 3 on the Build & Continuous Integration aspect. Second was Level 2 (29%), Levels 1 and 4 scored equal at 15%. For Quality Assurance, the two largest groups are Level 3 (38%) and Level 2 (28%). 18% of the participants voted on Level 1, the remaining 15% on Level 4. On the Visibility & Reporting aspect, the largest group is Level 2 (41%). The remaining respondents rated their organisation Level 3 (27%),

Level 1 (20%), and Level 4 (13%). On the Technology & Architecture aspect, 40% of the respondents assessed their organisation to be on Level 3 and 29% on Level 2. The remaining respondents estimated to be either on Level 4 (21%) or Level 1 (10%).

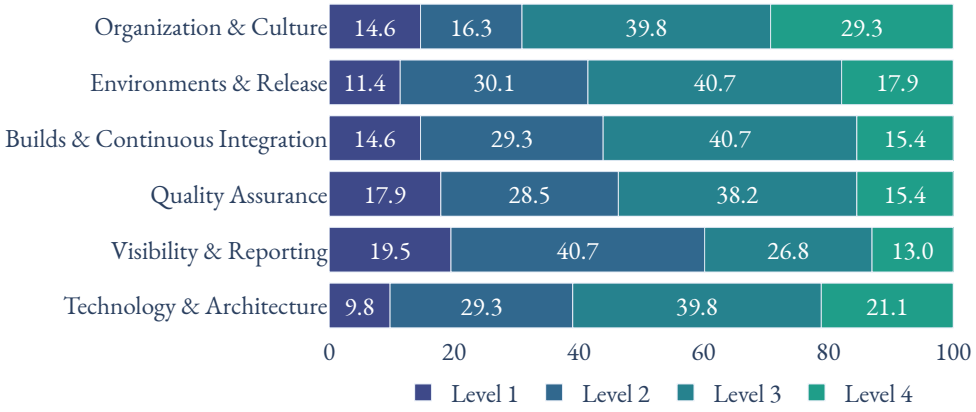


Figure 4.6: *DevOps maturity across six aspects.*

As can be seen in Figure 4.7, on the Portfolio level, the largest group of respondents estimated their organisation to be on the Beginner level (32%), followed by the Fluent level (26%). Only a small percentage of the respondents (7%) assessed their organisation to be World-class on this level. The remaining participants considered themselves either Novice (23%) or Advanced (13%). On the Program level, the two largest groups of participants rated their organisation Novice (28%) or Beginner (27%). The other participants assessed it to be Fluent (20%), Advanced (16%), and World-class (8%). For the Team level, the largest groups are Novice (33%) and Fluent (29%). The other participants estimated they were either Beginner (16%), Advanced (13%), or World-class (9%). The average maturities from Portfolio to Team are 48.0, 50.1, and 53.2, respectively.

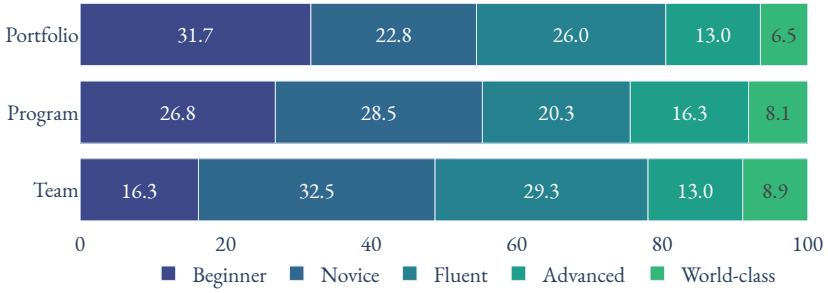


Figure 4.7: *Agile maturity across three levels.*

4.3 DevOps Practices & Tools

This section takes a look at the practices and tools respondents follow and use when developing, testing, and deploying software developed using DevOps concepts. It also explores the day-to-day activities of the respondents.

Figure 4.8 shows how often respondents apply the most common DevOps practices, as seen in Table 2.1. The practices are ranked based on their adoption rate, which was determined by following the algorithm of Serban et al. [71]. The most commonly implemented practice is 'everything as code and stored under version control', with 87.0% of the participants using it most of the time or more. Only 0.8% of the respondents never follow this practice. Other common practices that are applied most of the time or more are: 'automated and continuous monitoring' (76.4%), 'automated dashboards' (78.1%), 'trying to reduce time to test/QA' (71.1%), and 'change-based code-reviews' (71.5%). The practice that is implemented the least among the respondents is 'sandboxes for minimum code deployment', with 53.7% of the respondents either applying to it sometimes or never. Two other less applied practices involve test-driven and trunk-based development.

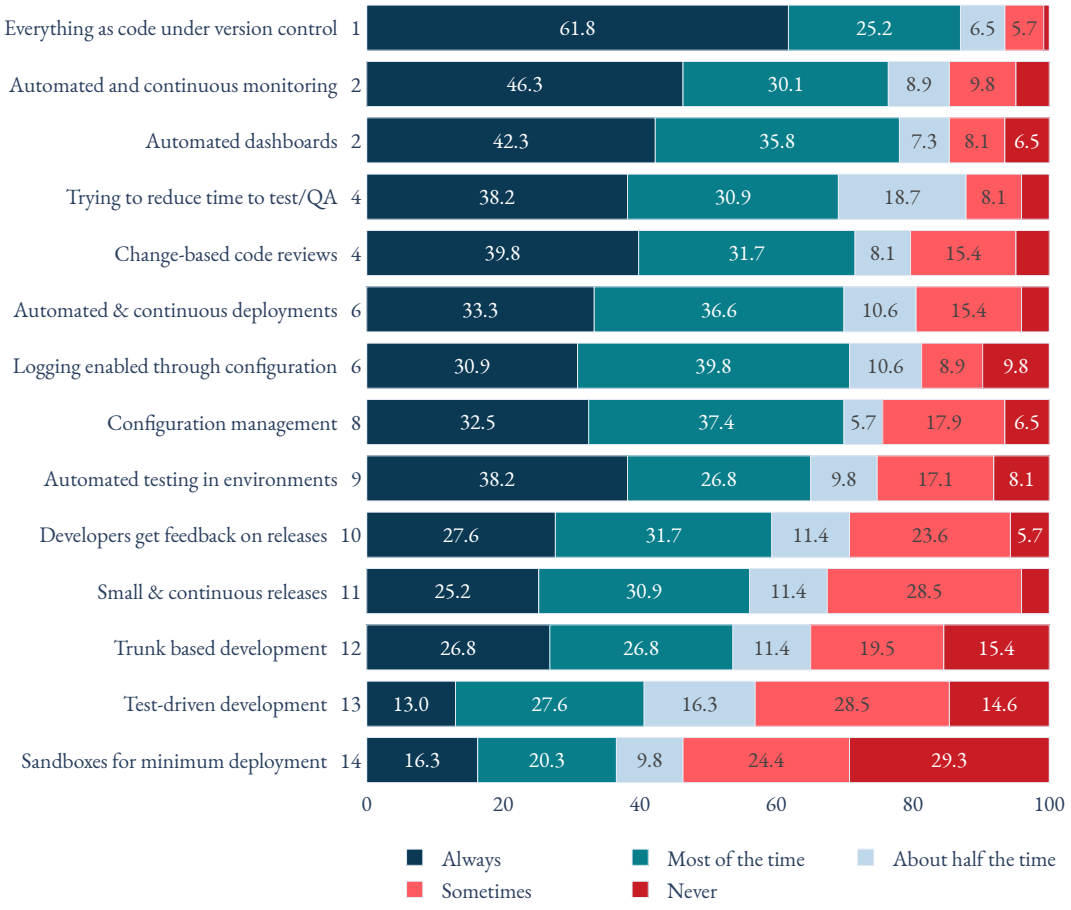


Figure 4.8: *Usage of DevOps practices.*

Table 4.4 contains the five most used DevOps tools for each practice, corresponding to the practices in Figure 4.8. Tools in the categories continuous delivery, continuous deployment, and continuous integration are often used for all three practices, with Azure DevOps being the most popular tool. Azure DevOps also ranks fourth in the version control management category. Microsoft's Azure platform is included five times in the figure, the GitLab ecosystem four times. The tools in the remaining categories focus solely on performing that task, with Azure Application Insights being the

only exception. All DevOps tools used by the participants can be found in Appendix G. There, the combination of tools is left unchanged.

Table 4.4: *Five most used DevOps tools per practice.*

Practice	Tool #1	Tool #2	Tool #3	Tool #4	Tool #5
Continuous delivery	Jenkins (61)	Azure DevOps (18)	GitLab CI (16)	Bamboo (11)	Visual Studio App Center (11)
Continuous deployment	Azure DevOps (16)	GitLab CI (14)	CodeDeploy (14)	Bamboo (7)	Jenkins (6)
Continuous integration	Jenkins (53)	Azure DevOps (20)	GitLab CI (16)	Bamboo (8)	CircleCI (5)
Version control management	GitHub (51)	Bitbucket (36)	GitLab (31)	Azure DevOps (22)	Nexus (12)
Configuration management	Ansible (52)	Puppet (16)	Chef (9)	Windows Server IIS7 (7)	Terraform (6)
Testing	Selenium (47)	Junit (35)	Jmeter (29)	Cucumber (26)	Cypress (8)
Code analysis	SonarQube (67)	SonarLint (19)	Jacoco (14)	JS hint (10)	CheckStyle (8)
Monitoring	Splunk (25)	Prometheus (17)	Nagios (12)	New Relic (11)	Graphite (11)
Logging	Kibana (39)	Logstash (38)	Graylog (7)	Azure Application Insights (5)	Datadog (5)

To get a better understanding of the DevOps pipeline, respondents were asked to describe the contents of the main DevOps pipeline. The results can be found in Figure 4.9. Provisioning, in this figure, also included deployment to a testing environment. Build (75.4%), unit tests (69.7%), and provisioning (66.4%) are the most common processes found in the pipeline. Integrations with monitoring (52.5%) and chatbots (50.0%) are used in half of the respondent's organisations. Security (37.7%) and performance (36.1%) tests are used the least.

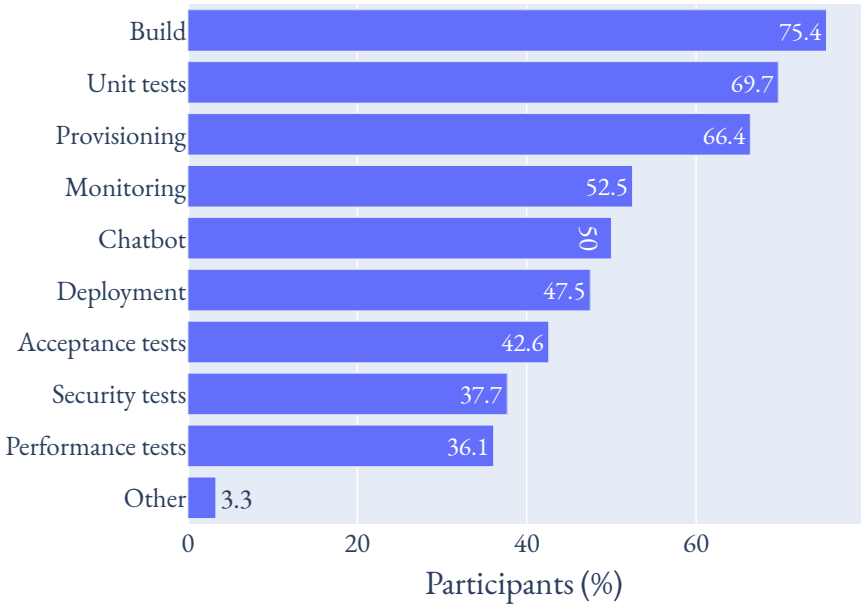


Figure 4.9: *Contents of the main DevOps pipeline.*

4.3.1 How Do Respondents Divide Their Time?

The survey contained three questions regarding capacity allocation: measuring how respondents spent their time. The first question asked respondents to divide their time into four categories. Figure 4.10 shows the results: most respondents spend their time developing features (39.3%), followed by maintenance (22.8%). Technical debt (19.2%) and enablers (18.7%) are the remaining categories. When these categories are averaged across DevOps and Agile maturity (Appendix F), we can see that the percentage of Features and Enablers increases when maturity increases. On the contrary, time spent on Technical Debt and Maintenance decreases when maturity increases.

Figure 4.11 depicts a different allocation of time: business features versus improving infrastructure. The average respondent spends about 50% of their time on both. This number does not change when comparing by maturity.

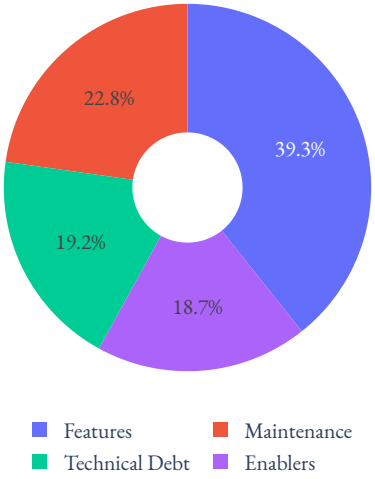


Figure 4.10: *Time spent working across four aspects.*

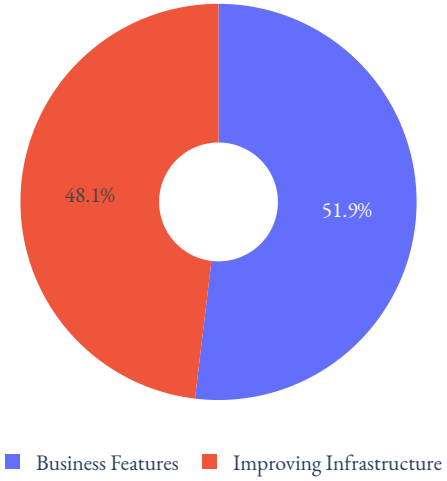


Figure 4.11: *Time spent on Business Features vs Improving Infrastructure.*

The last metric regarding time allocation is unplanned work. On average, respondents spend 34.0% of their time working on unplanned work.

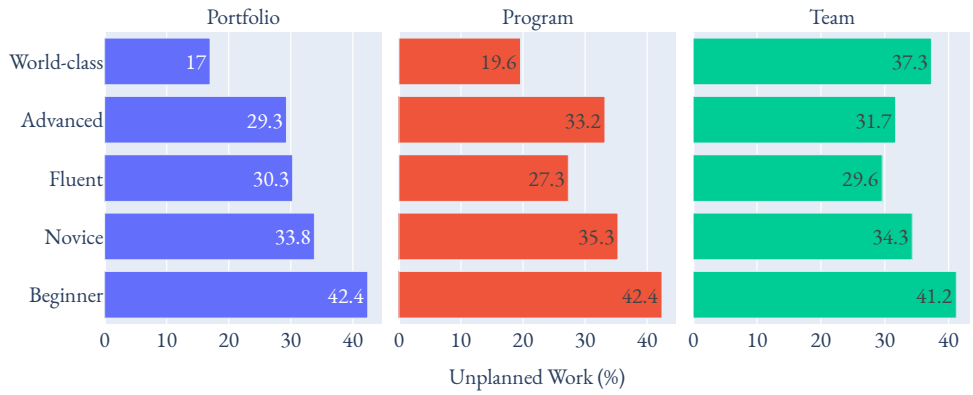


Figure 4.12: *Unplanned work across three Agile layers.*

When unplanned work is plotted for each layer and level of the Agile maturity model, we get the results as shown in Figure 4.12. On the Portfolio and Program layer,

unplanned work decreases when maturity increases. However, on the Team layer, the trend is more U-shaped; unplanned work decreases up until the fluent level, then increases again. Figure 4.13 shows unplanned work for each DevOps maturity layer. Across all layers, unplanned work decreases when maturity increases. Hence, there is a negative correlation between DevOps maturity and unplanned work.

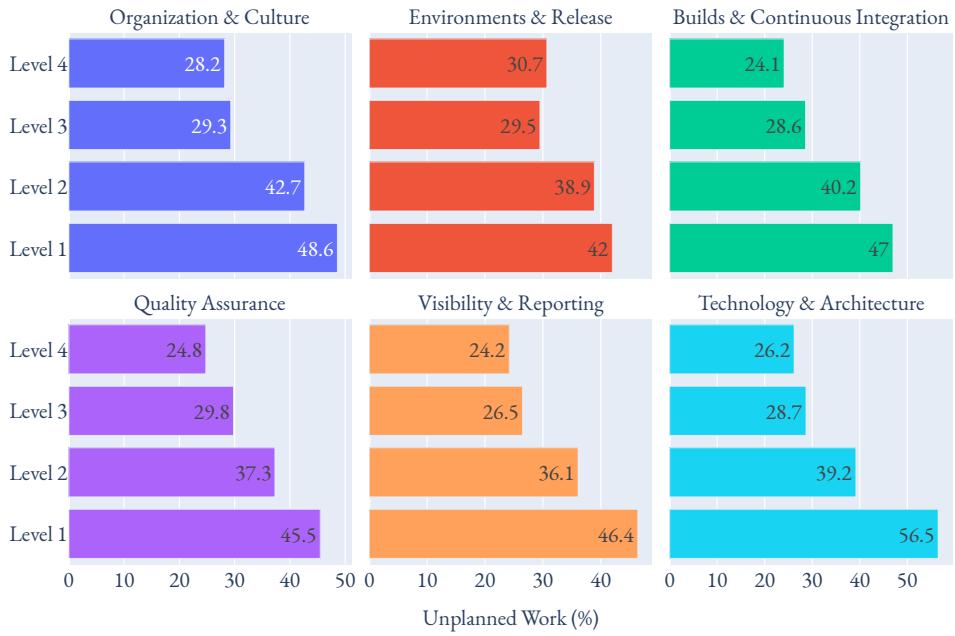


Figure 4.13: *Unplanned work across six DevOps layers.*

4.3.2 Hosting & Deployment

Figures 4.14 and 4.15 show the methods respondents use to deploy and host their – through DevOps developed – software. Close to 75% of the respondents use containerisation or virtualisation to deploy their software. Functions or Platforms as a Service are less popular: 7% and 6%, respectively. Ten percent uses a combination of multiple methods, often client dependent.

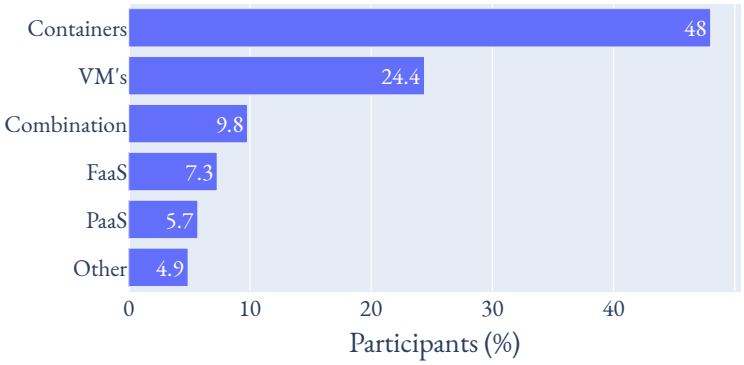


Figure 4.14: *Deployment methods.*

Public clouds are the most popular option for hosting (31.7%), followed by hybrid

clouds (26.0%) – a combination of public and private clouds. On-premise data centres are used in 14.6% of the organisations, private clouds in 13.0% of the organisations.

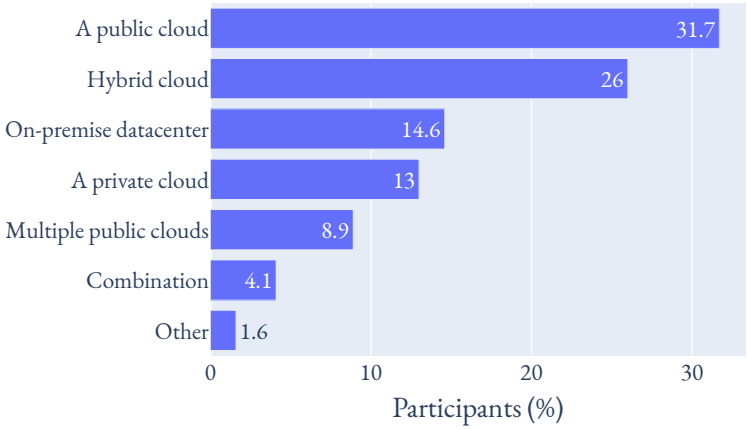


Figure 4.15: *Hosting methods.*

4.3.3 Usage of Open Source Software

Figure 4.16 shows the usage of open source software across the respondents. Most respondents (84.6%) agree to make extensive use of open source software. A neutral response is given by less than one-tenth (8.9%) of the respondents. A small fraction of the respondents (1.6%) makes no use of open source software. The industry in which most people agreed to making extensive use of open source software is the Retail, Consumer & E-commerce industry. Ninety percent of the respondents in this sector answered with 'Strongly agree'. The lowest usages of open source software are in the Other (22.2%) and Education (33.3%) sectors.

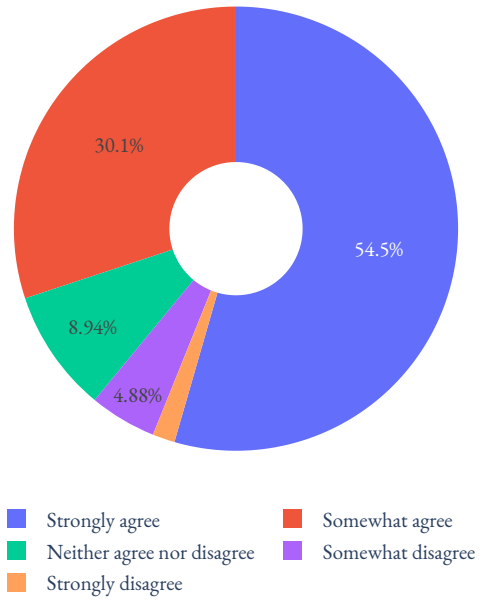


Figure 4.16: *Open Source usage.*

4.3.4 Results of Integration Tests

Figure 4.17 depicts how quickly developers can see the results of integration tests. The largest groups of respondents can see their results between one and ten minutes (35.0%), between ten minutes and one hour (26.8%), and between one hour and one day (15.4%). 10.6% of the participants can see the results less than one minute after running them. None of the respondents have to wait more than one month on the results of the integration tests. However, 6.5% of the respondents do not have any testing in place.

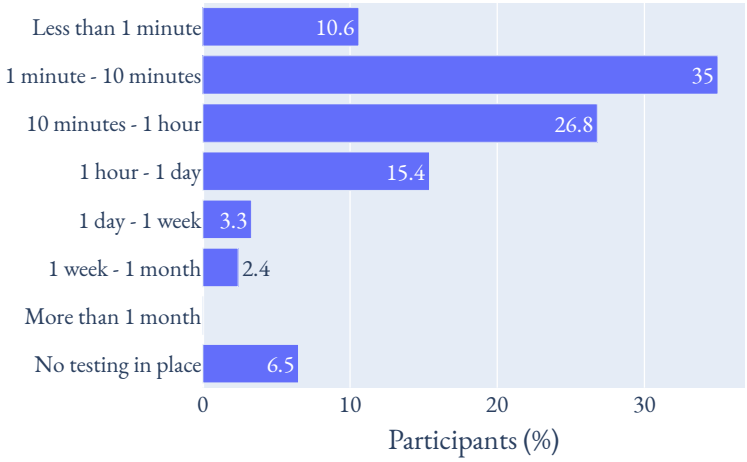


Figure 4.17: *Time before developers can see the results of integration tests.*

4.3.5 Customer Satisfaction

The most popular methods for measuring customer satisfaction are Net Promoter Score – or NPS for short – (15.3%), Surveys (14.5%), and direct feedback (10.5%). Direct conversations with customers (4.0%) and measuring complaints via customer support (5.6%) are less common methods participants use. Other (34.7%) measurement methods are only used by a single respondent, such as: sales, number of website visitors, or yearly/monthly metrics. About one-sixth of the respondents (16.1%) do not actively measure customer satisfaction. Some reasons include: software is not yet in production, software is for internal use only, or it is not the responsibility of the respondent.

4.4 Impact & Performance

Figure 4.18 displays the average organisational impact of a DevOps implementation across thirteen aspects and grouped into six dimensions (based on the research of Stettina et al. [72]). On average, all thirteen aspects show a positive impact. However, for all impact dimensions, respondents have experienced negative impacts. For both 'makes work more fun' and 'increases predictability of product delivery', 25% of the respondents experienced a negative impact ranging from -100 to 0. The aspects with the highest perceived impact are: 'improves time to market' (47.6%), 'increases collaboration' (47.0%), 'enables the earlier detection of defects' (45.8%), and 'increases the

effectiveness of development' (45.0%). The impact dimensions with the lowest average impact are: 'increases the usefulness of the product' (29.4%), 'increases the usability of the product' (30.5%), 'makes work less hectic' (30.8%), and 'makes work more planned' (31.4%). Industry wise, the average impact of Insurance (23.8%) and Government (26.2%) was the lowest. Healthcare, Pharma & Life sciences had the biggest impact (75.4%), followed by Education (62.3%) and Financial Services (53.0%).

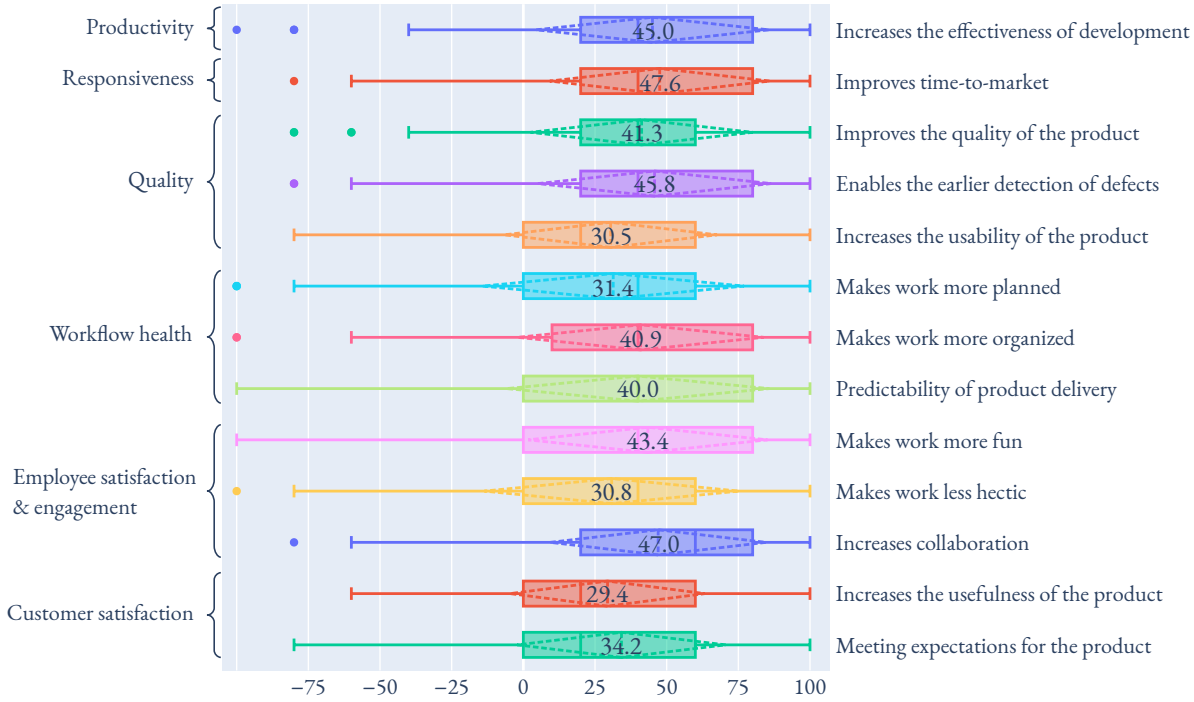


Figure 4.18: *DevOps impact on thirteen aspects (as from [69]).*

Figure 4.19 depicts organisational performance across six aspects. The two largest groups across all aspects are 'Performed above goals' and 'Met goals', with 'Performed above goals' being the largest of the two. These groups combined account for 50% to 60% of all respondents. On each aspect, at least 50% of the respondents performed slightly above goals or better and more than 80% of the respondents met their goals or better. For each performance metric, less than 15% of the respondents rated their organisation to perform below goals or well below goals. Very few participants (2% or less) performed well below goals. Participants performed best on the Profitability aspect, with only 8.5% of the respondents not meeting their goals. On the performance metrics Effeciency and Achieving Goals, respondents performed the worst, with 14.6% of the respondents not meeting goals. Similar to impact, the Healthcare, Pharma & Life Sciences, and Education industries also had the biggest average performance. The industries with the lowest recorded average performance are Insurance and Government.

4.4.1 Software Delivery Performance

Earlier research by Forsgren et al. [21] has shown that software delivery performance is related to organisational performance. For this reason, and to validate their work, measures for software delivery performance have been included in the survey.

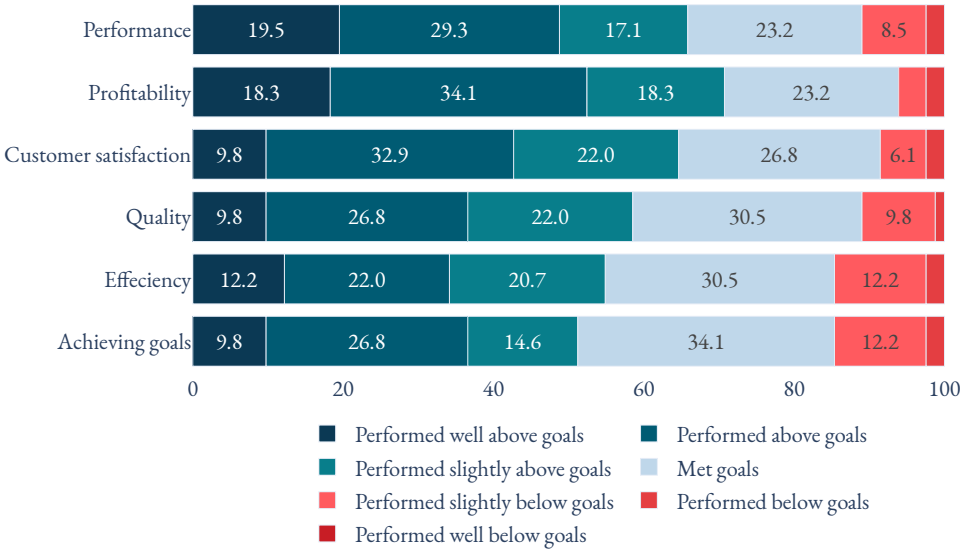


Figure 4.19: *Organisational performance across six aspects (as from [70]).*

Figure 4.20 shows how often participants deploy code for their primary service or application. Most respondents (37.4%) deploy code between once per week and once per month. The next two largest groups deploy on demand (24.3%) and between once per day and once per week (20.9%). The remaining participants are equally divided (both 8.7%) into 'between once per hour and once per day' and 'between once per month and once every six months'. None of the participants deploys fewer than once every six months.

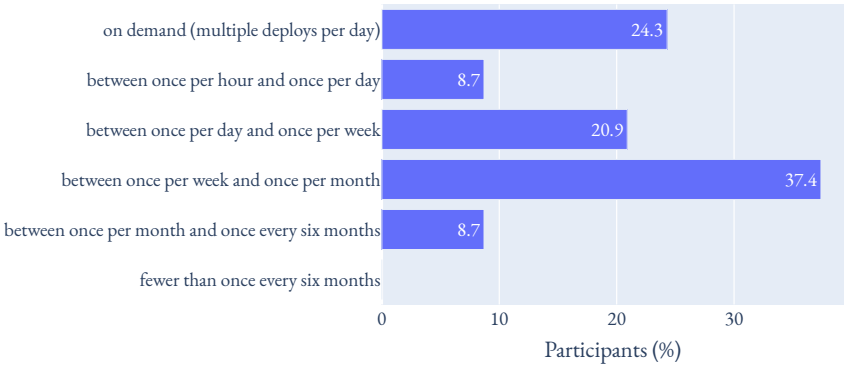


Figure 4.20: *Deployment frequency for primary service or application.*

Figure 4.21 depicts how much time it takes to go from code committed to code successfully running in production. For most respondents (33.6%) this takes less than one hour. A small percentage (0.9%) of the respondents waits more than six months to see their work in production. The remaining participants assessed the time as follows: less than one day (18.1%), between one day and one week (19.8%), between one week and one month (22.4%), and between one month and six months (5.2%).

Figure 4.22 depicts how long it generally takes to restore service (rollbacks included) for the primary service or application when a service incident occurs. The two largest groups of respondents can restore incidents within one hour (47.4%) or one day (45.7%). The remaining participants need between one day and one week (5.2%), be-

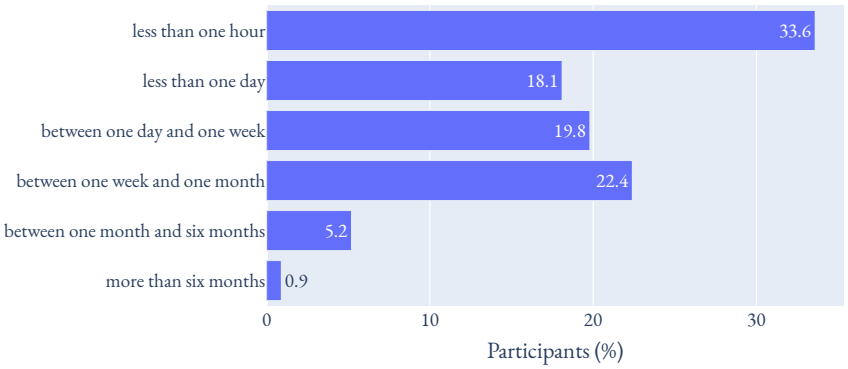


Figure 4.21: *Time from code committed to running in production.*

tween one week and one month (0.9%), or between one month and six months (0.9%). None of the respondents requires more than six months to solve incidents.

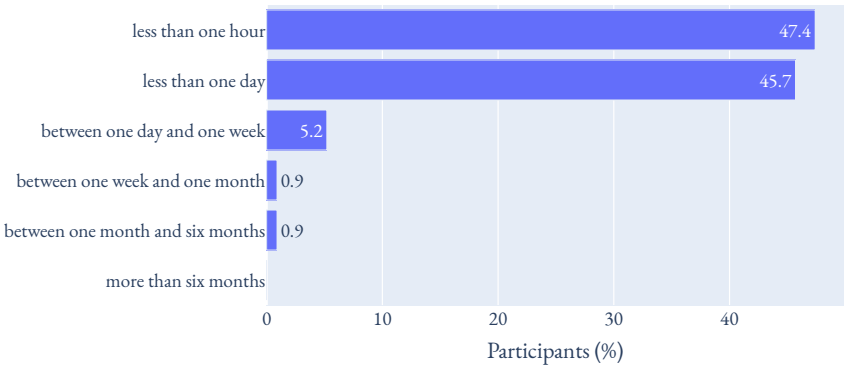


Figure 4.22: *Mean time to restore service after incidents.*

Figure 4.23 shows what percentage of changes for the primary service or application results in degraded service or requires remediation. Most respondents (42.2%) have to remediate between one and five percent of changes. The second largest group of participants has to fix less than one percent (21.1%) of changes, closely followed by 6% – 15% (18.9%). A small percentage of participants (6.6%) has to remediate 31% of changes or more.

4.5 Correlations & Cluster Analysis

To make informed recommendations to DevOps practitioners, first a clear understanding of the statistics behind the data is required. In this section correlations between the aforementioned tools, practices, maturities and performance metrics are explored.

4.5.1 Correlations

Figure 4.24 shows the correlation between DevOps maturity and practice usage. The lowest correlating practices are 'sandboxes for minimum code deployment', 'logging enabled through configuration', and 'trunk-based development'. Stongly correlating

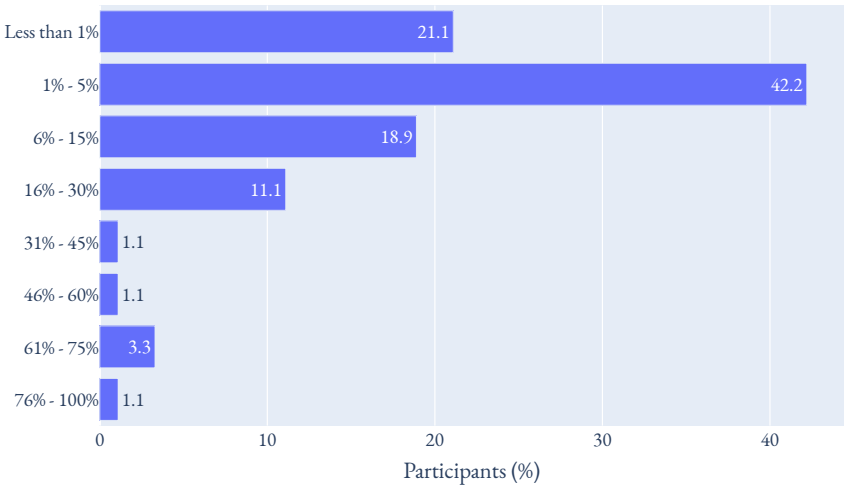


Figure 4.23: *Percentage of changes requiring remediation.*

practices include 'automated and continuous testing in development and staging environments', 'automated and continuous deployments', and 'automated and continuous monitoring'. The highest overall correlation is between 'automated testing in environments' and Quality Assurance (0.499), the lowest correlation can be found on 'sandboxes for minimum code deployment' and Quality Assurance (0.005).

	O&C	E&R	B&CI	QA	V&R	T&A
Automated & continuous deployments	0.281**	0.34***	0.378***	0.295***	0.373***	0.276**
Small & continuous releases	0.187*	0.283**	0.309***	0.25**	0.353***	0.318***
Developers get feedback on releases	0.117	0.159	0.247**	0.182*	0.237**	0.16
Sandboxes for minimum deployment	0.093	0.013	0.114	0.005	0.102	0.102
Everything as code under version control	0.2*	0.138	0.255**	0.15	0.241**	0.33***
Configuration management	0.206*	0.276**	0.334***	0.382***	0.314***	0.35***
Automated testing in environments	0.252**	0.309***	0.481***	0.499***	0.43***	0.326***
Trying to reduce time to test/QA	0.139	0.074	0.199*	0.202*	0.246**	0.273**
Change-based code reviews	0.264**	0.161	0.213*	0.253**	0.294***	0.228*
Automated and continuous monitoring	0.327***	0.364***	0.321***	0.267**	0.316***	0.246**
Automated dashboards	0.353***	0.271**	0.243**	0.19*	0.318***	0.248**
Logging enabled through configuration	0.274**	0.168	0.195*	0.195*	0.201*	0.264**
Trunk based development	0.079	0.182*	0.16	0.077	0.04	0.177
Test-driven development	0.158	0.156	0.211*	0.287**	0.342***	0.109

Figure 4.24: *Correlation between DevOps maturity and practices.*

Figure 4.25 depicts the correlation between Agile maturity and practice usage. Practices with high correlations for all Agile layers are 'automated and continuous testing in development and staging environments', 'small and continuous releases', and 'automated and continuous deployments'. Likewise, the lowest correlating practices are 'sandboxes for minimum code deployment', 'logging enabled through configuration',

and 'trunk-based development'. The highest overall correlation is between Portfolio and 'small and continuous releases', the lowest correlation between 'sandboxes for minimum code deployment' and Program or Team.

	Portfolio	Program	Team
Automated & continuous deployments	0.381***	0.364***	0.235**
Small & continuous releases	0.401***	0.392***	0.279**
Developers get feedback on releases	0.277**	0.302***	0.181*
Sandboxes for minimum deployment	0.146	0.125	0.125
Everything as code under version control	0.275**	0.23*	0.167
Configuration management	0.265**	0.283**	0.305***
Automated testing in environments	0.391***	0.381***	0.383***
Trying to reduce time to test/QA	0.22*	0.175	0.277**
Change-based code reviews	0.282**	0.227*	0.274**
Automated and continuous monitoring	0.294***	0.328***	0.372***
Automated dashboards	0.33***	0.355***	0.264**
Logging enabled through configuration	0.216*	0.187*	0.149
Trunk based development	0.217*	0.185*	0.203*
Test-driven development	0.398***	0.326***	0.278**

Figure 4.25: *Correlation between Agile maturity and practices.*

Figure 4.26 illustrates the correlation between Agile and DevOps maturity and (software delivery) performance. The first four metrics on the y-axis are the software delivery performance metrics described in Section 4.4.1. The remaining six metrics are the performance metrics presented by Widener [70]. The metrics 'Customer satisfaction', 'Quality', and 'Restoration time' show the strongest correlations across both maturities. The 'Remediation %', 'Performance', and 'Profitability' metrics show the weakest correlations. The strongest correlation is between 'Customer satisfaction' and 'Quality Assurance', the weakest correlation between 'Remediation %' and 'Team'.

Figures 4.27 and 4.28 show the correlation between DevOps practices and performance or software delivery performance, respectively. The practice of 'automated and continuous deployments' as well as 'small and continuous releases' and 'everything as code under version control' have the strongest correlations with the performance metrics. The practice of 'change-based code reviews' has the weakest (even negative) correlations. The performance metric 'Effeciency' correlates strongly with all DevOps practices, except for 'change-based code reviews' and 'trying to reduce time to test/QA'. Again, 'Profitability' is among the weakest correlating metrics.

For software delivery performance, 'Restoration time' and 'Remediation %' show the weakest correlations with DevOps practices. The practice of 'small and continuous releases' correlates strongly with both 'Deployment frequency' and 'Deployment time'. The DevOps practice of 'automated and continuous monitoring' correlates the most with all software delivery performance metrics. The practices 'test-driven development', 'configuration management', and 'logging enabled through configuration' show little correlation with all metrics.

	Portfolio	Program	Team	O&C	E&R	B&CI	QA	V&R	T&A
Deployment frequency	0.251*	0.133	0.336**	0.044	-0.016	-0.007	0.078	0.191	0.036
Deployment time	0.229	0.14	0.276*	0.017	0.217	0.113	0.07	0.253*	0.067
Restoration time	0.192	0.172	0.205	0.26*	0.35**	0.3*	0.304*	0.313**	0.361**
Remediation %	0.111	-0.033	-0.195	-0.03	0.081	0.078	0.092	0.101	0.136
Performance	0.165	0.125	0.144	0.06	0.05	0.075	0.107	0.159	0.065
Profitability	0.133	0.168	0.205	0.093	0.17	0.116	0.147	0.191	0.095
Customer satisfaction	0.309*	0.33**	0.176	0.281*	0.214	0.336**	0.444***	0.253*	0.287*
Quality	0.313**	0.307*	0.147	0.26*	0.171	0.244*	0.334**	0.226	0.253*
Effeciency	0.267*	0.227	0.111	0.126	0.102	0.088	0.206	0.199	0.109
Achieving goals	0.371**	0.283*	0.2	0.159	0.171	0.136	0.206	0.229	0.117

Figure 4.26: *Correlation between Agile and DevOps maturity and performance.*

	Performance	Profitability	Customer satisfaction	Quality	Effeciency	Achieving goals
Automated & continuous deployments	0.294**	0.248*	0.257*	0.319**	0.482***	0.341**
Small & continuous releases	0.306**	0.197	0.26*	0.344**	0.177	0.389***
Developers get feedback on releases	0.181	0.135	0.1	0.25*	0.244*	0.149
Sandboxes for minimum deployment	0.148	0.15	0.218*	0.255*	0.292**	0.157
Everything as code under version control	0.326**	0.277*	0.196	0.253*	0.323**	0.193
Configuration management	-0.034	-0.017	0.253*	0.298**	0.371***	0.148
Automated testing in environments	0.178	0.124	0.253*	0.243*	0.281*	0.249*
Trying to reduce time to test/QA	0.229*	0.268*	0.228*	0.329**	0.087	0.236*
Change-based code reviews	-0.01	0.08	-0.105	-0.042	-0.111	-0.101
Automated and continuous monitoring	0.142	0.172	0.105	0.182	0.146	0.196
Automated dashboards	0.087	0.143	0.157	0.087	0.174	0.142
Logging enabled through configuration	0.1	0.132	0.219*	0.275*	0.229*	0.187
Trunk based development	0.144	0.152	0.149	0.208	0.319**	0.128
Test-driven development	0.213	0.26*	0.281*	0.195	0.392***	0.175

Figure 4.27: *Correlation between DevOps practices and organisational performance.*

Figure 4.29 depicts the correlation between software delivery performance and organisational performance. The 'Deployment frequency' metric correlates the strongest with all organisational performance metrics, 'Remediation %' shows the weakest correlation across all performance metrics. The performance metrics 'Achieving goals' and 'Customer satisfaction' shows the strongest correlations across three of the four software delivery performance metrics.

	Deployment frequency	Deployment time	Restoration time	Remediation %
Automated & continuous deployments	0.244*	0.243*	0.158	0.11
Small & continuous releases	0.508***	0.479***	0.18	0.224*
Developers get feedback on releases	0.165	0.233*	-0.03	0.127
Sandboxes for minimum deployment	0.295**	0.132	0.038	0.041
Everything as code under version control	0.271*	0.148	0.125	0.223*
Configuration management	0.104	0.018	-0.003	0.074
Automated testing in environments	0.059	0.087	0.323**	0.132
Trying to reduce time to test/QA	0.285**	0.127	0.027	0.06
Change-based code reviews	0.143	0.187	0.254*	0.164
Automated and continuous monitoring	0.325**	0.351***	0.276**	0.257*
Automated dashboards	0.238*	0.237*	0.14	0.218*
Logging enabled through configuration	-0.001	-0.067	0.106	0.143
Trunk based development	0.157	0.19	0.074	0.093
Test-driven development	0.167	0.105	-0.127	-0.092

Figure 4.28: *Correlation between DevOps practices and software delivery performance.*

	Deployment frequency	Deployment time	Restoration time	Remediation %
Performance	0.396***	0.257*	0.035	0.001
Profitability	0.313**	0.181	0.105	-0.067
Customer satisfaction	0.338**	0.182	0.255*	0.038
Quality	0.295*	0.071	0.082	0.045
Effeciency	0.211	0.147	-0.004	-0.012
Achieving goals	0.342**	0.204	0.292*	0.006

Figure 4.29: *Correlation between software delivery performance and organisational performance.*

4.5.2 Clustering

Figure 4.30 shows the result of cluster analysis on the four metrics of software delivery performance and displayed on the average performance (y-axis) and average software delivery performance (x-axis). Two distinct clusters can be identified: one with high average performance, and one with medium average performance.

Table 4.5 lists the average practice usage for both clusters. Although the practice 'logging enabled through configuration' is used equally across the two groups, for almost all practices, high performers apply them more frequently. High performers are 1.4 times more likely to implement the practice of 'small & continuous releases', the biggest difference between the groups. In addition, high performers are 1.5 times more likely to use containerisation. The differences in usage of open source software

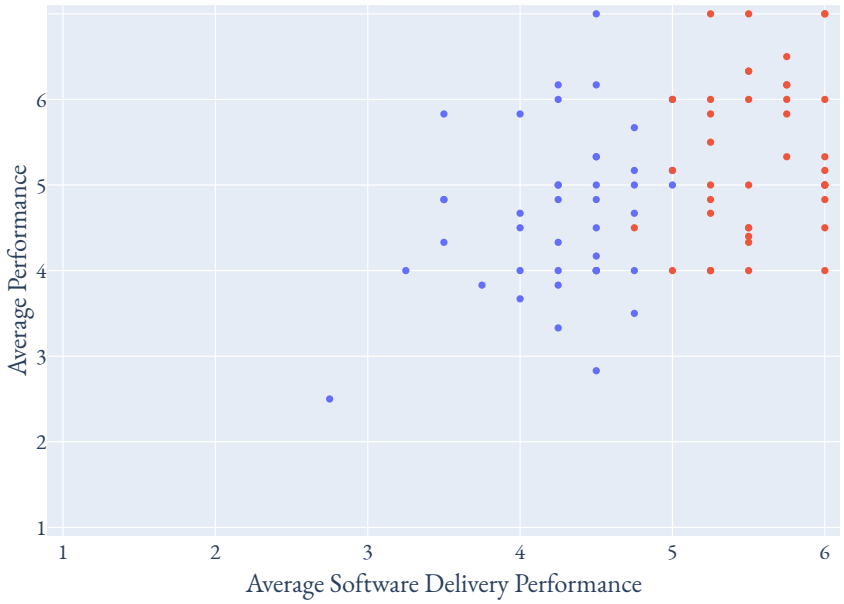


Figure 4.30: *Clustering based on software delivery performance.*

is insignificant.

Table 4.5: *Average practice usage by medium and high performers.*

Practice	Medium Performers	High Performers
Automated & continuous deployments	3.7	4.1
Small & continuous releases	3.0	4.2
Developers get feedback on releases	3.4	4.0
Sandboxes for minimum deployment	2.5	3.0
Everything as code under version control	4.4	4.7
Configuration management	3.8	3.9
Automated testing in environments	3.7	4.0
Trying to reduce time to test/QA	4.0	4.3
Change-based code reviews	3.9	4.4
Automated and continuous monitoring	3.8	4.6
Automated dashboards	3.9	4.4
Logging enabled through configuration	3.9	3.9
Trunk based development	3.2	3.7
Test-driven development	2.9	3.1

Table 4.6 contains the top five tools for both medium and high performers. For continuous integration, delivery, and deployment, high performers use Azure DevOps and Jenkins less often and Bamboo and GitLab CI more often than the medium performers. The same can be seen in the version control management tools. The testing tools show little difference between the groups. High performers also use Datadog for both monitoring and logging, which is never used by the medium performers.

Table 4.6: *Top five tools for medium and high performers.*

Tool	Medium Performers	High Performers
Continuous delivery	Jenkins (27)	Jenkins (16)
	Visual Studio App Center (6)	GitLab CI (9)
	Azure DevOps (6)	Bamboo (8)
	TeamCity (3)	Azure DevOps (4)
	GitLab CI (3)	TeamCity (3)
Continuous deployment	CodeDeploy (7)	GitLab CI (9)
	Azure DevOps (6)	CodeDeploy (7)
	Octopus Deploy (4)	Bamboo (6)
	Travis CI (3)	Azure DevOps (4)
	Ansible (2)	Travis CI (2)
Continuous integration	Jenkins (24)	Jenkins (13)
	Azure DevOps (9)	GitLab CI (8)
	GitLab CI (4)	Bamboo (6)
	Codeship (3)	Azure DevOps (3)
	CircleCI (2)	CircleCI (2)
Version control management	GitHub (17)	GitHub (21)
	Bitbucket (13)	GitLab (12)
	Azure DevOps (9)	Bitbucket (11)
	GitLab (9)	Azure DevOps (4)
	Nexus (6)	Artifactory (4)
Configuration management	Ansible (17)	Ansible (20)
	Puppet (9)	Terraform (5)
	Chef (4)	Chef (4)
	Windows Server IIS7 (3)	Puppet (3)
	Azure App Configuration (1)	SaltStack (2)
Testing	Selenium (17)	Selenium (19)
	Junit (14)	Jmeter (13)
	Jmeter (10)	Junit (11)
	Cucumber (8)	Cucumber (9)
	TestNG (3)	Jest (3)
Code analysis	SonarQube (27)	SonarQube (20)
	SonarLint (8)	SonarLint (7)
	Jacoco (7)	JS hint (4)
	FindBugs (4)	CheckStyle (4)
	CheckStyle (3)	Snyk (3)
Monitoring	Splunk (12)	Prometheus (9)
	Prometheus (6)	Datadog (8)
	Graphite (5)	Splunk (8)
	Nagios (5)	New Relic (5)
	New Relic (4)	Elastic Stack (4)
Logging	Kibana (19)	Logstash (15)
	Logstash (12)	Kibana (11)
	Azure Application Insights (3)	Datadog (4)
	Elastic Search (2)	Graylog (4)
	Sumo Logic (2)	Log.io (2)

Chapter 5

Discussion & Recommendations

This chapter focusses on interpreting the results of Chapter 4 and gives recommendations for practitioners. To organise and ensure all findings are being discussed properly, the chapter is organised in three different sections.

5.1 Impact of DevOps Practices and Tools on Organisational Performance

Looking at the collected data, the four most applied DevOps practices are: 1) everything stored as code and under version control, 2) automated and continuous monitoring, 3) automated dashboards, and 4) trying to reduce time to test/QA. Half of those practices pertain to the automated (and continuous) monitoring of products or services. This could be due to two reasons. First, most cloud providers enable monitoring and logging functionalities by default [5]. By choosing a cloud provider for (a part of) the DevOps pipeline, monitoring is automatically included. Second, within the CALMS and SAFe framework, measurement is considered a key activity. With about one-fifth of the respondents using CALMS and a quarter using SAFe to introduce DevOps and Agile into organisations, measurement could have high priority. These monitoring practices have little impact on organisational performance, but do impact deployment frequency and deployment time significantly. The practice of 'automated and continuous monitoring' has strong correlations with DevOps maturity. Therefore, this practice, which seems many organisations can implement, can be used to improve other aspects of organisations' DevOps maturity.

The most popular practice being 'everything stored as code and under version control' is not unsurprising. As literature already mentioned, this practices has many benefits [13, 21, 25], such as increased transparency, rollbacks in case of emergency, and quickly setting up architecture. Moreover, it also the key practice in achieving a fully automated pipeline [21, 30]. If the practice of 'automated and continuous deployments' was split up into two practices, the automated part would potentially have a resembling adherence as the most popular one. Storing everything as code and under version control correlates strongly with almost all metrics for organisational performance, especially performance and efficiency. The same is true for the practice of 'automated and continuous deployments'.

Another practice that correlates strongly with both organisational performance and software delivery performance is 'small and continuous releases'. Moreover, this practices also positively impacts both maturities. Next, it is used 1.4 times as much by high performers than their medium scoring colleagues. Investing time and resources into following this practices thus might be a good first place to start.

The practices that have been implemented the least are as follows: 1) sandboxes for minimum code deployment, 2) test-driven development, and 3) trunk-based development. These practices also do not show a strong correlation with most performance metrics. DevOps is thus not so much about which development method is used, but rather how rapid (via automation) software can be delivered. However, these development focussed practices do have a positive impact on efficiency, which might help increase lead time.

The least implemented DevOps practice of using sandboxes for minimum deployment also has little impact on both DevOps and Agile maturity, as well as most metrics for organisational performance and software delivery performance. Two notable exceptions are efficiency (0.292***) and deployment frequency (0.296***). Other practices such as 'automated and continuous deployment' and 'small and continuous releases', however, have a higher impact on those metrics as well as high correlations with both maturities. Even though there is little evidence this practice impacts performance, high performers do use it 1.2 times more than medium performers.

5.1.1 Bamboo Solely Used By High Performers

For the first three practices – continuous delivery, deployment, and integration – a set of similar tools are used (e.g. Azure DevOps, Jenkins, or Bamboo). Moreover, some of those tools are also used for version control management. Respondents who use Azure DevOps, for example, are likely to use this tool for all four practices. This is in contrast with the reasoning of Kersten [43], who mentioned DevOps practitioners demanding specialised tools with limited functionality. In practice, we thus see these *platform* providers offering similar services based on the continuous integration and delivery of software, including version control.

The remaining practices show little overlap of tools. This can be expected, especially for testing and code analysis: these tools are often based on the programming language that is being used for development. Some overlap of tools does happen between the monitoring and logging practices. Again showing DevOps practitioners picking tools that offer more than one specialised feature, which is not consistent with Kersten's work.

Since the survey data does not allow for correlation of tools with performance, they cannot compared as easily as with practices. However, tools used by medium and high performers can be compared. For continuous integration, delivery, and deployment, high performers use Azure DevOps and Jenkins less often than medium performers. Bamboo and GitLab CI are used more often for these practices. In fact, Bamboo is only being used by high performers. The same yields for version control management tools. Next, for the logging and monitoring practices, high performers use Datadog eight and four times, respectively. Medium performers do not use this tool at all. So, does using Bamboo and Datadog improve the (software delivery) performance of organisations? Maybe. There is no certainty that switching to these tools will increase performance.

Perhaps future work could provide these insights.

5.1.2 DevOps & Agile Maturity

DevOps and Agile maturity are closely related; the correlations between the different layers range from 0.4*** to 0.71***. The Organisation & Culture and Technology & Architecture are considered to be on the highest levels. This first layer, furthermore, has the highest score on Level 4 (29%) of all layers. This is quite surprising as multiple scholars [3, 7, 13–15] have identified the shift in culture as one of the main challenges hindering the implementation and adoption of DevOps. Moreover, Level 4 requires frequent communication – another challenge – to share information, and even continuous communication with IT operators. Although literature has identified these challenges, DevOps practitioners have found ways to combat them and reach the highest levels of maturity. One explanation could be that the identification of these common challenges by researchers, allows practitioners to look for signs and pro-actively counter them as soon as they occur.

Visibility & Reporting is the least mature layer, followed by Quality Assurance. Moreover, the Visibility & Reporting layer has the most respondents on Level 1 (20%) and Level 2 (41%) of all layers. Subsequently, the least participants assessed their organisation to be on Level 3 and Level 4 for this layer. One reason for this could be that the Eficode model [38] has serious requirements for Level 3 and 4. To reach Level 3, "the status of requirements can be monitored in real-time in relation to tests and released features". Level 4 requires using metrics collected in development for improvements. While logging and monitoring are considering DevOps practices [4, 9, 22, 32], it is understandable that organisations might want to focus on other aspects first.

For Agile maturity, respondents rate their organisations lower scores when compared to DevOps maturity. For the Portfolio and Program layer, the Beginner groups are among the largest one. On the Team layer, however, organisations seem to be more mature: most respondents are among the Novice or Fluent levels. A reason for this could be that Agile transformations commonly use the bottom-up (team-by-team) strategy [72]. Agile is thus first introduced on a team level, before transforming the rest of the organisation. A takeaway for some of the participants of the reflective workshops was to carefully consider at which levels to apply Agile, as the value might be lower than expected.

For all levels, the Advanced and World-class are the smallest groups. However, participants of all reflective workshops indicated that 7 – 9% of the total population seemed quite high. They were very impressed, especially for the World-class teams, as the model requires Advanced and better to release zero errors to production environments.

Unplanned Work

For all six DevOps maturity layers, unplanned work decreases as maturity increases. Hence, there is a negative correlation between DevOps maturity and unplanned work. For some layers (e.g. Technology & Architecture, Builds & Continuous Integration, and Visibility & Reporting) the amount of unplanned work decreases with almost 50%

when transitioning from Level 1 to Level 4. As for the impact (see Figure 4.18), DevOps shows it makes work more planned by an average of 31.4%.

For Agile Maturity, however, only the Portfolio layer shows a clear decrease of unplanned work when maturity increases. On the Program layer, Advanced is the exception: it increases instead of showing a decrease. One of the *requirements* to be on the Advanced level is "continuous positive feedback from customers from last deliveries" [69]. It is possible this could lead to unplanned work, especially in the early phase when not all feedback might be positive. When this is tackled, however, organisations can be considered World-class and are able to respond to complex customer requests.

Time Allocation

Figure F.1 and Figure F.2 (see Appendix F) depict capacity allocation grouped by Agile and DevOps maturity, respectively. The four capacity aspects are: features, enablers, technical debt, and maintenance. The general consensus across the two figures is an increase in features and a decrease in technical debt and maintenance when maturity increases. More mature organisations thus spent more time developing features than less mature organisations. Similarly, less mature organisations spent more time on technical debt and maintenance. Time spent on enablers does not change significantly when maturity rises.

For Agile maturity specifically, the difference on the feature aspect between Beginner and World-class can be as much as 20% (Portfolio and Program layer). For the program level, this makes sense when looking at the maturity model used. Laanti's [69] requirements for World-class include the "ability to rapidly respond to customer needs". To be able to achieve this, most time should be spent on developing features. The same yields for the Portfolio layer: features play an important role in supporting continuous innovation and business development. Moreover, analysis* shows that work on this layer becomes more organised and planned as maturity increases.

For DevOps Maturity, the increase in features is not as straightforward. Most layers require a *boost* on Level 1 and Level 2 to technical debt and maintenance before developing features starts to increase. Some of these layers include: Technology & Architecture, Environments & Release, and Builds & Continuous Integration. This makes sense: to be able to leverage the advantages of continuously delivering features, first a stable architecture, easy-to-setup development environments, and a well-defined pipeline are required. When this is complete (Level 3), the focus shifts to development.

Improving Maturity

Now that the benefits of high DevOps and Agile maturity are clear, how can organisations reach these high levels? When looking at Figure 4.24 and Figure 4.25, DevOps practices that strongly correlate with the maturities can be identified. These practices can then be used to improve maturity. As discussed, DevOps and Agile maturity are closely related. In addition, practices that show a strong correlation with Agile maturity, also have a strong correlation with DevOps maturity. High correlating practices include: 1) automated and continuous testing in development and staging environments, 2) automated and continuous deployments, and 3) automated and continuous

Correlations are 0.11 and 0.19, respectively.

monitoring. Hence, these practices should be included in the first steps to improve both DevOps and Agile maturity of organisations. Then, organisations should decide on which maturity aspects they want to focus next, and follow DevOps practices accordingly.

Interestingly, while the Visibility & Reporting layer of the DevOps maturity model was reported to be the lowest, the practice of 'automated and continuous monitoring' has high correlations across all layers, including the Agile ones. So while it does not seem to be attractive to focus time and resources on this aspect, it may be worthwhile after all. The practice of 'automated and continuous deployments' being among the highest correlating practices is in accordance with the general consensus of DevOps.

Simply implementing the practices will not improve maturity. Many of the practices require a change in architecture, development methodology, or even culture. As seen, to reach higher levels of maturity, often a boost in technical debt is required. This investment can be worth it, however, as reaching high levels of DevOps and Agile maturity have shown to reduce unplanned work and allow for a focus on feature development. Moreover, most respondents show high levels of DevOps maturity on the Organisation & Culture and Technology & Architecture layers. This could mean that applying certain practices might be easier for some organisations as a solid technological foundation is already in place.

5.2 Technology Trends Fuelling DevOps

This section discusses the following two operational aspects of DevOps: containerisation and cloud computing. In addition, open source usage will be discussed.

5.2.1 Containerisation

Nearly half (48%) of the survey participants use containers to deploy their through DevOps developed software. This phenomenon explains why the most popular practice is 'everything as code under version control': to be able to containerise applications and services, configuration should be stored as well.

The 2018 State of DevOps report [65] found that users of containers are between 1.3 and 1.5 times more likely to be "elite" performers (software delivery performance is used as metrics for performance). This is similar to the difference between the high and medium performers in this research: high performers tend to use containerisation 1.5 times more often than medium performers. The 2021 State of DevOps report also measured the usage of containers and virtual machines. They too found containers (64%) and VM's (48%) to be among the most popular deployment options. The exact numbers cannot be compared since their research allowed respondents to choose multiple answers. Compared to the 2018 study [65], the usage of containers has more than doubled (from 31%). The increase in popularity thus reflects a transformation towards modern technologies [65]. Indeed, Spafford and Herschmann [55] found that the demand for containers keeps increasing. Container Management is therefore included in the Gartner Hype Cycle for Agile and DevOps [55]. The continuous demand is, according to Spafford and Herschmann, due to the "preference for container runtimes and packaging formats".

Another possible explanation for the large use of containerisation is that containers fit well within the most common architecture used in continuous delivery: microservices. Containers provide an easy way to create independent and autonomous environments. Services such as databases, business logic, and customer-facing front-ends can be separated. This also allows for teams to focus on a specific aspect of the entire system. Updates can be released independent of each other, possibly improving frequency.

To further explore the significant gap in deployment options, different performance metrics have been compared between two groups of respondents: those who use containers for deployment and those who do not. The results can be found in Table 5.1. For all metrics (albeit slightly for some) containers have a higher rating than no containers.

Table 5.1: *Differences in performance when comparing container usage.*

Metric	Containers	No Containers
Deployment frequency	4.25	3.87
Deployment time	4.89	4.0
Mean time to restore	5.5	5.31
Changes requiring remediation	5.75	5.51
Average impact	40.1	38.0
Average performance	5.0	4.9

Aside from benefits found in literature, such as ease of portability between development and production environments [45] and simplifying the overall architecture [55], containerisation will improve (software delivery) performance, analogous to the results of Forsgren et al. [65].

5.2.2 Cloud Computing

Respondents favour public and hybrid (a mix of public and private) clouds over other options, such as on-premise data centres. Ebert et al. [13] suggest DevOps and microservices "should be cloud-based as much as possible".

Table 5.2 shows cloud usage (in percentages) over the last years [65–67], the 2020 State of DevOps report did not include results on cloud usage. The State of DevOps reports have no clear definition of hybrid clouds, therefore the comparison with our finding (26%) cannot be made. The fact that cloud usage increases over time could thus also be explained by a different definition being used. In 2019, both public clouds and on-premise data centres are significantly higher than other years. The reports do not attempt to explain this difference.

Table 5.2: *Cloud usage in percentages of the last years (from [65–67]).*

Type	2018	2019	2021
Public	39	50	35
Private	32	23	29
Hybrid	18	27	34
No cloud	17	44	21
Multiple public	54	-	21

Although the exact numbers cannot be compared with this research – due to forcing respondents to chose only one answer – the same trend can be seen. Public clouds remain the most popular choice, followed by hybrid, and private clouds. Although cloud computing becomes more popular, a significant portion of organisation still use on-premise data centres.

Mohammad [73] sees advantages for cloud-based DevOps, such as: improved automation, increased accessibility, and better back-up management. Another benefit of cloud-based services over on-premise servers is increased flexibility of capacity. As customer demand changes, cloud applications can be scaled accordingly without the need for extra physical space. This allows for the virtualisation of resources being used in real-time [74]. Cost plannings can thus be more accurate.

Another reason cloud-based environments are favoured over traditional options is that cloud solutions work well with containers. Kubernetes, a popular open-source container orchestration tool, is readily available on all major cloud vendors, lowering the barrier to enter. In addition, vendors are responsible for the correct installation of Kubernetes. Cluster operators can thus focus solely on their tasks.

5.2.3 Open Source Usage

Section 4.3.3 mentioned the extensive use of open source software across the respondents. The results from this research are resemblent to the 2018 State of DevOps report [65]. There, 58% agreed to making extensive use of open source software, compared to 54.5% reported in the previous chapter. The highest performers in their study are 1.75 times more likely to use open source software and 1.5 times more likely to increase usage.

Kersten [43] mentions that DevOps and open source share a similar focus on "empowering the practitioner". With the increasing complexity of software development and delivery, practitioners demanded specialised tools, transforming the pipeline from a few tools that could do everything to many specialised tools. A benefit of using open source software within an organisation is that new hires are already familiar with the tools and technologies used [67].

Since the survey did not include questions regarding the type of product/service being developed, the open source usage cannot be divided into its two main categories: infrastructure and libraries. However, with the aforementioned popularity of containers (and the microservice infrastructure), the wide usage of open source infrastructure software – such as Docker or Kubernetes – could be explained. Moreover, popular tools for version control management (GitLab), continuous integration (Jenkins), and testing (Selenium) are also open source.

An explanation for why the Retail, Consumer, & E-commerce industry contains the largest group of open source users, is that virtually all web frameworks are open source. The biggest JavaScript framework for the web – e.g. React, Angular, NextJS, or NodeJS – are all licensed under the MIT license.

5.3 Comparing Agile and DevOps Survey Data

Within this section, the outcomes of this research are compared to earlier scientific studies.

5.3.1 DevOps Attracts Different Practitioners

Stettina and colleagues [72] conducted a study on the impact of agile transformations on organisational performance. Similar to this research, they too used a survey and measured Agile maturity. Figure 5.1 displays the Agile maturity from their research.

Before the results can be compared, first the difference in demographics has to be addressed. The different focus of the studies results in different target groups: Agile Program Coaches (26.1%), Trans-formation Leads (21.6%), and Team Coaches (21.6%) versus Software Developers (30.1%), IT Operations Engineers (22.0%), and Automation Engineers (8.9%). In addition, the most common size of organisations in Stettina's data set is larger: 50,000+ (38.1%) and 1,001 - 5,000 (23,9%) compared to 20 - 99 (24.0%) and 100 - 499 (20.7%). While using similar channels to distribute the survey, this study has attracted a different group of practitioners than Stettina's work.

Figure 5.1: *Agile maturity across three levels [72].*

The overall (average) maturity compared to Stettina's work is lower on the Program and Team level, but higher on the Portfolio level. Visually comparing Figure 4.7 and Figure 5.1 shows the following: 1) world-class remains the same for all levels, 2) advanced decreased on all levels in favour of fluent, and 3) most beginner and novice levels are decreasing compared to the earlier study. The trend thus seems to be a growing maturity across all levels.

The differences between Stettina's study and this research could be due to the differences in the target group: Agile and Team coaches might rank themselves higher than operational employees such as developers and engineers. An explanation for the decrease in advanced (and increase of fluent) for all levels could be that organisations completed an Agile transformation (thinking quite highly of themselves) and are now introducing DevOps into the organisation. They have come to realise that their earlier assessment of maturity may have been a little too high. The decrease in beginner and novice levels is somewhat expected: as time passes, organisations should continue to improve their Agile maturity. The overall increase in maturity on the Portfolio level could be due to COVID-19. Many organisations are forced to enable their employees to work from home, requiring them to continuously focus on portfolio decision-making and data collection.

5.3.2 Similar Impact Trends; Different Values

The study by Stettina et al. [72] also included some of the same impact dimensions used in this research. Comparing the two shows the following. Although the overall impact is lower in this research, the trends remain the same. The impact on 'makes work less hectic' is the lowest in both studies, followed by 'makes work more planned'. The highest impacts are on 'increases collaboration' (an important aspect in both Agile and DevOps), 'improves time-to-market', and 'enables the earlier detection of defects'.

There are three reasons why the overall impact could be lower than in 2018. First, the target groups of the studies are different. Agile coaches and trainers might evaluate the impact of Agile higher than developers or engineers. Second, the range of possible answers differs between the studies. In the 2018 study [72] a range from 0 - 100 was used, meaning all impact is positive. Third, the pandemic forced many people to work from home. This could have an impact on e.g. performance, planning, or efficiency. However, since the difference is almost 20% for every metric, the most likely explanation is the change in range, reducing respondents' bias.

5.3.3 Software Delivery Performance & Organisational Performance

Forsgren et al. [33] studied the impact of software delivery on organisational performance. They found that their latent construct for software delivery performance (consisting of delivery frequency, delivery time, MTTR, and remediation %) positively relates to organisational performance. This research included their latent construct as well as the commonly used metrics for organisational performance [70].

As can be seen in Figure 4.29, only the deployment frequency shows a strong correlation with the organisational performance metrics. Deployment time shows little correlation, the remaining metrics even include negative correlations. A reason for this could be that their research includes data from a larger number of respondents, compared to the 123 participants in this study.

Nevertheless, deployment frequency (and to some extent deployment time) is a good predictor of organisational performance. Therefore, aforementioned practices such as 'automated and continuous deployments', 'small and continuous releases', and 'everything stored as code and under version control' are DevOps practices that can favorably impact organisational performance.

5.4 Limitations

Since this research is based on a web-based survey, some types of bias can occur.

The first bias is related to sampling: the way respondents are chosen to participate. Two roles (Software Developer/Engineer and IT Operations/Infrastructure Engineer) are responsible for more than 50% of the respondents. This might have lead to self-selection bias: respondents selecting themselves to participate in the research. Self-selection bias was combatted by sharing the survey in different online communities, consisting of different roles and nationalities.

The second type of bias is non-response bias: people not participating in the study differ significantly from those who do, resulting in a under-representation. Since the

response rate of the survey was 36.5%, this might have been the case. The input of managers or coaches, for example, takes up less than 15% of the respondents. Their responses could differ significantly from DevOps practitioners, and can be a good addition for future work.

Another limitation is the lack of similar research on this topic. Although many scholars have given input on the best DevOps practices, their input sometimes originated from single case studies. Although based on relevance and occurrence, the practices chosen for the survey could have a misalignment between theory and practice. Future work could repeat the research with different practices and tools.

Chapter 6

Conclusion

This study focussed on the effects of DevOps practices and tools on organisational performance. The main research question, therefore, was: 'what are the effects of DevOps practices and technologies on organisational performance?'. Using an anonymous web-based survey, data from 123 respondents was gathered. Respondents were targeted through private channels, online communities on, for example, LinkedIn, and direct messages to practitioners.

The most common roles of the respondents were Software Developer/Engineer, IT Operations/Infrastructure Engineer, and Automation Engineer/Expert. Most of the organisations (85.4%) are currently undergoing or have recently undergone either a DevOps or Agile transformation. Most transformations are estimated to be completed between six months and two years. The CALMS DevOps framework is the most popular one practitioners use. Since the Scaled Agile Framework also includes a strategy for DevOps, 37.9% of the respondents who used SAFe, did not use a separate DevOps framework.

Commonly implemented DevOps practices include: 1) everything as code and under version control, 2) automated and continuous monitoring, 3) automated dashboards, and 4) trying to reduce time to test/QA. The least popular practices are 'sandboxes for minimum code deployment' and two regarding development: trunk-based or test-driven development. DevOps is thus not so much about which development method is used, but rather how rapid (via automation) software can be delivered and monitored.

Popular DevOps tools include Azure DevOps, Jenkins, GitLab, and Bamboo. Bamboo and Datadog are solely used by high performers, indicating there might be some advantage to using these tools. High performers also tend to make more use of GitLab CI, but use Azure DevOps and Jenkins less often. While literature suggest that tooling becomes more specialised, the data from this research shows that tools in the CI/CD categories can be used for four practices: continuous deployment, delivery, and integration, as well as version control management. A minimal deployment pipeline is thus possible by selecting a single (open-source) platform.

Other technologies, such as containerisation and open-source software have shown to fuel DevOps. Containers are used 1.5 times more often by high performers than their medium scoring colleagues. These results are similar to the work of an earlier conducted study. While the usage of open source usage did not differ significantly

within this research, previous research related open source usage with better software delivery performance.

DevOps and Agile maturity are closely related. Improving maturity has its benefits: as maturity increases, respondents spent less time on unplanned work and maintenance. Instead, they are able to develop new features. For DevOps maturity specifically, an initial *boost* on maintenance and technical debt is observed. To leverage the advantages of continuously delivering features, a solid technical foundation is thus required. To improve maturity, I recommend the following three practices: 1) automated and continuous testing in development and staging environments, 2) automated and continuous deployments, and 3) automated and continuous monitoring.

Comparing the results of this work to previous studies reveals the following. First, while using a similar distribution strategy, DevOps triggers different practitioners to respond to surveys than an Agile topic did. Moreover, Agile maturity has increased between the two studies. Next, the impact of DevOps and Agile show similar trends. Both mindsets increase collaboration, improve time-to-market, and enable the earlier detection of defects.

After unravelling the mysteries around DevOps, its practices and tools, the overall conclusion of this research is as follows. Applying practices that improve lead time (e.g. continuous and automated deployments or everything as code and under version control) will have a positive impact on both software delivery performance and organisational performance. In addition, certain tools (Bamboo and Datadog) are only used by high performers. More general technologies, such as containerisation and open source have also shown a positive impact on organisational performance.

For DevOps practitioners, the practices 'automated and continuous deployments', 'small and continuous releases', and 'everything stored as code and under version control' have shown to positively impact both software delivery and organisational performance. Although implementation might require an overhaul of existing architectures, the effort could be worthwhile.

6.1 Future Work

As mentioned several times in this research, the survey did not include questions regarding the type of product or service that is being developed. Further research could incorporate this question and give recommendations specific for that kind of product or service. This would also allow for a more detailed comparison for both DevOps and Agile maturity.

Next, future work could repeat distributing the survey and 1) verify if this research is accurate, but more importantly 2) see if organisations start to improve on both DevOps and Agile maturity as well as software delivery performance and organisational performance by applying the recommended practices.

While this research aimed to provide a recommended toolchain, the data did not support this. Although two tools were only used by high performers, it remains unclear whether the tools are solely responsible. Future work could thus focus on a more specific set of tools or compare organisations who use these tools. A case study, for example, could compare two teams within an organisation while working on similar products, but differ in the DevOps tools they use.

Lastly, future work could dive deeper into the usage of open source software. As seen, open source usage has been adopted by most of the respondents. It remains unclear, however, how open source software is used. This research did not differentiate between, for example, open source architecture (e.g. Docker of Kubernetes), libraries (e.g. ReactJS or Angular), or other tooling. It would be interesting to further investigate the usage and impact of open source software in DevOps.

Bibliography

[1] L. Zhu, L. Bass, and G. Champlin-Scharff, *DevOps and its Practices*, IEEE Software **33**, 32 (2016).

[2] S. Sharma and B. Coyne, *DevOps for Dummies*, John Wiley & Sons, 2015.

[3] J. Smeds, K. Nybom, and I. Porres, *DevOps: A Definition and Perceived Adoption Impediments*, in *Agile Processes in Software Engineering and Extreme Programming*, pages 166–177, Cham, 2015, Springer International Publishing.

[4] R. Jabbari, N. bin Ali, K. Petersen, and B. Tanveer, *What is DevOps? A Systematic Mapping Study on Definitions and Practices*, in *Proceedings of the Scientific Workshop Proceedings of XP2016*, Association for Computing Machinery, 2016.

[5] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, *A survey of DevOps concepts and challenges*, ACM Computing Surveys (CSUR) **52**, 1 (2019).

[6] M. Callanan and A. Spillane, *DevOps: making it easy to do the right thing*, Ieee Software **33**, 53 (2016).

[7] L. Riungu-Kalliosaari, S. Mäkinen, L. E. Lwakatare, J. Tiihonen, and T. Männistö, *DevOps Adoption Benefits and Challenges in Practice: A Case Study*, in *International conference on product-focused software process improvement*, pages 590–597, Springer, 2016.

[8] J. Hamunen et al., *Challenges in Adopting a Devops Approach to Software Development and Operation*, (2016).

[9] L. E. Lwakatare, T. Kilamo, T. Karvonen, T. Sauvola, V. Heikkilä, J. Itkonen, P. Kuvaja, T. Mikkonen, M. Oivo, and C. Lassenius, *DevOps in Practice: A Multiple Case study of Five Companies*, Information and Software Technology **114**, 217 (2019).

[10] C. Technologies, *TechInsights Report: What Smart Businesses Know About DevOps*, Technical report, CA Technologies, 2013.

[11] M. Senapathi, J. Buchan, and H. Osman, *DevOps Capabilities, Practices, and Challenges: Insights from a Case Study*, in *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, pages 57–67, 2018.

[12] M. Shahin, M. Zahedi, M. A. Babar, and L. Zhu, *Adopting Continuous Delivery and Deployment: Impacts on Team Structures, Collaboration and Responsibilities*, in *Proceedings of the 21st international conference on evaluation and assessment in software engineering*, pages 384–393, 2017.

[13] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, *DevOps*, IEEE Software **33**, 94 (2016).

[14] A. Wiedemann, N. Forsgren, M. Wiesche, H. Gewald, and H. Krcmar, *Research for practice: the DevOps phenomenon*, Communications of the ACM **62**, 44 (2019).

[15] G. Bou Ghantous and A. Gill, *DevOps: Concepts, Practices, Tools, Benefits and Challenges*, PACIS2017 (2017).

[16] S. Overflow, *Developer Survey Results 2018*, `https://insights.stackoverflow.com/survey/2018`, 2017, [Online; accessed 03/10/2021].

[17] S. Overflow, *Developer Survey Results 2017*, `https://insights.stackoverflow.com/survey/2017`, 2017, [Online; accessed 03/10/2021].

[18] M. Azure, *DevOps vs. agile*, `https://azure.microsoft.com/en-us/overview/devops-vs-agile/`, [Online; accessed 08/11/2021].

[19] T. Hall, *Agile vs. DevOps*, `https://www.atlassian.com/devops/what-is-devops/agile-vs-devops`, [Online; accessed 08/11/2021].

[20] A. Brunnert et al., *Performance-oriented DevOps: A research agenda*, arXiv preprint arXiv:1508.04752 (2015).

[21] N. Forsgren, J. Humble, and G. Kim, *Accelerate: The science of lean software and devops: Building and scaling high performing technology organizations*, IT Revolution, 2018.

[22] A. Balalaie, A. Heydarnoori, and P. Jamshidi, *Microservices Architecture Enables DevOps: An Experience Report on Migration to a Cloud-Native Architecture*, Ieee Software **33**, 42 (2016).

[23] F. M. Erich, C. Amrit, and M. Daneva, *A Qualitative Study of DevOps Usage in Practice*, Journal of software: Evolution and Process **29**, e1885 (2017).

[24] N. Forsgren, D. Smith, J. Humble, and J. Frazelle, *2019 Accelerate State of DevOps Report*, Technical report, Google, 2019.

[25] J. Humble and J. Molesky, *Why Enterprises Must Adopt Devops to Enable Continuous Delivery*, Cutter IT Journal **24**, 6 (2011).

[26] E. van Ommeren, M. van Doorn, J. Dial, and D. van Herpen, *Design to Disrupt*.

[27] I. Buchanan, *CALMS Framework*, `https://www.atlassian.com/devops/frameworks/calms-framework`, [Online; accessed 07/09/2021].

[28] DevOpsGroup, *CALMS Model of DevOps*, `https://pages.devopsgroup.com/calms-model-of-devops`, [Online; accessed 22/11/2021].

[29] R. Wilsenach, *DevOpsCulture*, `https://martinfowler.com/bliki/DevOpsCulture.html`, 2015, [Online; accessed 07/09/2021].

[30] L. E. Lwakatare, P. Kuvaja, and M. Oivo, *An Exploratory Study of DevOps Extending the Dimensions of DevOps with Practices*, ICSEA 2016 **104** (2016).

[31] P. Rodríguez, A. Haghighatkhah, L. E. Lwakatare, S. Teppola, T. Suomalainen, J. Eskeli, T. Karvonen, P. Kuvaja, J. M. Verner, and M. Oivo, *Continuous deployment of software intensive products and services: A systematic mapping study*, Journal of Systems and Software **123**, 263 (2017).

[32] J. R. Hamilton et al., *On Designing and Deploying Internet-Scale Services*, in *LISA*, volume 18, pages 1–18, 2007.

[33] N. Forsgren, J. Humble, and G. Kim, *Accelerate: The science of lean software and devops: Building and scaling high performing technology organizations*, IT Revolution, 2018.

[34] M. K. Aljundi et al., *Tools and Practices to Enhance DevOps Core Values*, (2018).

[35] R. Vaasanthi, V. P. Kumari, and S. P. Kingston, *Analysis of Devops Tools using the Traditional Data Mining Techniques*, International Journal of Computer Applications **161**, 47 (2017).

[36] J. Scheuner, J. Cito, P. Leitner, and H. Gall, *Cloud Workbench: Benchmarking IaaS providers based on Infrastructure-as-Code*, in *Proceedings of the 24th International Conference on World Wide Web*, pages 239–242, 2015.

[37] A. A. Ur Rahman and L. Williams, *Security Practices in DevOps*, in *Proceedings of the Symposium and Bootcamp on the Science of Security*, pages 109–111, 2016.

[38] Eficode, *DevOps: Eficode Quick Guide*, Technical report, Eficode, 2020.

[39] M. Zarour, N. Alhammad, M. Alenezi, and K. Alsarayrah, *A research on DevOps maturity models*, International Journal of Recent Technology and Engineering **8**, 4854 (2019).

[40] Merriam-Webster, *Practice*, `https://www.merriam-webster.com/dictionary/practice`, [Online; accessed 25/09/2021].

[41] D. Stahl, T. Martensson, and J. Bosch, *Continuous Practices and DevOps: Beyond the Buzz, What Does It All Mean?*, in *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 440–448, IEEE, 2017.

[42] C. Patel and M. Ramachandran, *Agile maturity model (AMM): A Software Process Improvement framework for agile software development practices*, International Journal of Software Engineering, IJSE **2**, 3 (2009).

[43] M. Kersten, *A Cambrian Explosion of DevOps Tools*, IEEE Computer Architecture Letters **35**, 14 (2018).

[44] H. Akshaya, J. Vidya, and K. Veena, *A Basic Introduction to DevOps Tools*, International Journal of Computer Science & Information Technologies **6**, 05 (2015).

[45] M. Stillwell and J. G. Coutinho, *A DevOps Approach to Integration of Software Components in an EU Research Project*, in *Proceedings of the 1st International Workshop on Quality-Aware DevOps*, pages 1–6, 2015.

[46] A. Janes, V. Lenarduzzi, and A. C. Stan, *A Continuous Software Quality Monitoring Approach for Small and Medium Enterprises*, in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, pages 97–100, 2017.

[47] P. Austel, H. Chen, T. Mikalsen, I. Rouvellou, U. Sharma, I. Silva-Lepe, and R. Subramanian, *Continuous Delivery of Composite Solutions: A Case for Collaborative Software Defined PaaS Environments*, in *Proceedings of the 2nd International Workshop on Software-Defined Ecosystems*, pages 3–6, 2015.

[48] J. Wettinger, U. Breitenbücher, and F. Leymann, *Standards–based DevOps Automation and Integration Using TOSCA*, in *Proceedings of the 7th International Conference on Utility and Cloud Computing (UCC 2014)*, pages 59–68, IEEE Computer Society, 2014.

[49] S. Krusche and L. Alperowitz, *Introduction of Continuous Delivery in Multi-Customer Project Courses*, in *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 335–343, 2014.

[50] M. Guerriero, M. Ciavotta, G. Gibilisco, and D. Ardagna, *Space4cloud: a DevOps environment for multi-cloud applications*, in *Proceedings of the 1st International Workshop on Quality-Aware DevOps*, pages 29–30, 2015.

[51] M. Shahin, M. A. Babar, and L. Zhu, *The Intersection of Continuous Deployment and Architecting Process: Practitioners' Perspectives*, in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–10, 2016.

[52] H. Yasar and K. Kontostathis, *Where to integrate security practices on DevOps platform*, International Journal of Secure Software Engineering (IJSSE) **7**, 39 (2016).

[53] J. Roche, *Adopting DevOps practices in quality assurance*, Communications of the ACM **56**, 38 (2013).

[54] S. Newman, *Building microservices: designing fine-grained systems*, O'Reilly Media, Inc., 2015.

[55] G. Spafford and J. Herschmann, *Hype Cycle for Agile and DevOps, 2021*, Technical report, Gartner, 2021.

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

[57] M. C. Becker, *Organizational routines: a review of the literature*, Industrial and Corporate Change **13**, 643 (2004).

[58] G. Dosi et al., *The Nature and Dynamics of Organizational Capabilities*, Oxford university press, 2000.

[59] A. M. Knott, *The organizational routines factor market paradox*, Strategic Management Journal **24**, 929 (2003).

[60] D. J. Collis, *Research note: how valuable are organizational capabilities?*, Strategic management journal **15**, 143 (1994).

[61] D. J. Teece, G. Pisano, and A. Shuen, *Dynamic capabilities and strategic management*, Strategic management journal **18**, 509 (1997).

[62] C. Salvato, *Capabilities unveiled: The role of ordinary activities in the evolution of product development processes*, Organization Science **20**, 384 (2009).

[63] B. J. Biddle and D. S. Anderson, *Theory, methods, knowledge, and research on teaching*, Handbook of Research on Teaching **3**, 230 (1986).

[64] A. Joinson, *Social desirability, anonymity, and Internet-based questionnaires*, Behavior Research Methods, Instruments, & Computers **31**, 433 (1999).

[65] N. Forsgren, J. Humble, and G. Kim, *Accelerate: 2018 State of DevOps*, Technical report, Google, 2018.

[66] A. Brown, N. Kersten, and M. Stahnke, *2020 State of DevOps Report*, Technical report, Puppet, 2019.

[67] D. Smith, D. Villalba, M. Irvine, D. Stanke, and N. Harvey, *Accelerate: 2021 State of DevOps*, Technical report, Google, 2021.

[68] N. Forsgren and J. Humble, *DevOps: Profiles in ITSM performance and contributing factors*, Forsgren, N., J. Humble (2016)." DevOps: Profiles in ITSM Performance and Contributing Factors." In the Proceedings of the Western Decision Sciences Institute (WDSI) (2016).

[69] M. Laanti, *Agile transformation model for large software development organizations*, in *Proceedings of the XP2017 Scientific Workshops*, pages 1–5, 2017.

[70] S. K. Widener, *An empirical analysis of the levers of control framework*, Accounting, organizations and society **32**, 757 (2007).

[71] A. Serban, K. van der Blom, H. Hoos, and J. Visser, *Adoption and Effects of Software Engineering Best Practices in Machine Learning*, in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–12, 2020.

[72] C. J. Stettina, V. van Els, J. Croonenberg, and J. Visser, *The Impact of Agile Transformations on Organizational Performance: A Survey of Teams, Programs and Portfolios*, in *International Conference on Agile Software Development*, pages 86–102, Springer, Cham, 2021.

[73] S. M. Mohammad, *Streamlining DevOps automation for Cloud applications*, International Journal of Creative Research Thoughts , 2320 (2018).

[74] J. Wettinger, A. Vasilios, and F. Leymann, *Automated Capturing and Systematic Usage of DevOps Knowledge*, in *Proceedings of the IEEE International Conference on. IEEE Computer Society*, Citeseer, 2015.

[75] R. Penners and A. Dyck, *Release Engineering vs. DevOps - An Approach to Define Both Terms*, Full-scale Software Engineering , 49 (2015).

[76] F. Erich, C. Amrit, and M. Daneva, *Report: DevOps Literature Review*, University of Twente, Tech. Rep (2014).

Appendix A

Definitions of DevOps

Table A.1 displays all definitions of DevOps found in literature.

Table A.1: *DevOps definitions found in scientific literature.*

#	Definition	Reference
1	DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality.	[56]
2	DevOps efficiently integrates development, delivery, and operations, thus facilitating a lean connection of these traditionally separated silos.	[13]
3	DevOps ensures quick release cycles and promotes a collaborative, integrated communication platform which should be made available to all project stakeholders.	[52]
4	DevOps is a development methodology aimed at bridging the gap between Development (Dev) and Operations (Ops), emphasizing communication and collaboration, continuous integration, quality assurance, and delivery with automated deployment utilizing a set of development practices.	[4]
5	DevOps is a collaborative and multidisciplinary effort within an organization to automate continuous delivery of new software versions, while guaranteeing their correctness and reliability.	[5]
6	DevOps is a set of practices that is trying to bridge developer-operations gap at the core of things and at the same time covers all the aspects which help in speedy, optimized and high quality software delivery.	[15]
7	DevOps is a set of engineering process capabilities supported by certain cultural and technological enablers.	[3]
8	DevOps is a mindset substantiated with a set of practices to encourage cross-functional collaboration between teams - especially development and IT operations - within a software development organization, in order to operate resilient systems and accelerate delivery of change.	[30]

Continued on next page

Table A.1 – continued from previous page

#	Definition	Reference
9	DevOps is a mindset, encouraging cross-functional collaboration between teams - especially development and IT operations - within a software development organization, in order to operate resilient systems and accelerate delivery of changes.	[75]
10	DevOps is a conceptual framework for reintegrating development and operations of Information Systems.	[76]
11	DevOps is a set of principles and practices to improve collaboration between development and IT Operations.	[11]
12	DevOps is a practice in which operations and developers work together in each stage of DevOps life cycle (from development stage to production stage).	[44]
13	DevOps is an emerging paradigm to tightly integrate developers with operations personnel.	[48]
14	DevOps is a set of practices which not only aims to decrease the time between applying a change to a system and the change being transferred to the production environment, but also insists on keeping the software quality in terms of both code and the delivery mechanism as one of the key elements in the development process.	[22]
15	DevOps is a set of principles and practices for smoothing out the gap between development and operations in order to continuously deploy stable versions of an application system.	[20]
16	DevOps is an approach in which traditional software engineering roles are merged and communication is enhanced to improve the production release frequency and maintain software quality.	[7]
17	DevOps is an interaction between development and operations personnel on three levels: individuals, teams, and departments.	[23]

Appendix B

Practices Found in Literature

Table B.1 contains practices identified in scientific literature and combined where possible.

Table B.1: *Practices identified in scientific literature.*

#	Practice	Category	References
1	Architectures are designed for failure but without single points of failure	Architecture	[4, 32]
2	Components are decoupled and do not affect each other	Architecture	[32]
3	Components are restartable and redundant	Architecture	[32]
4	Integrated change management prepared for crisis situations	Change Management	[4, 76]
5	Version roll-back is supported	Compatibility	[32]
6	Maintain forward and backward compatibility	Compatibility	[32]
7	Everything is stored as code and under version control	Configuration	[4, 9, 15, 32, 33]
8	Integrated configuration management	Configuration	[4]
9	Continuous delivery	Continuous Delivery	[4, 11, 22, 33, 41]
10	Continuous integration	Continuous Integration	[4, 9, 33, 41, 42]
11	Production data is used to find problems in quality assurance and performance	Data	[4, 32]
12	Automated and continuous deployment throughout entire pipeline	Deployment	[4, 9, 11, 15, 33, 34, 41]

Continued on next page

Table B.1 – continued from previous page

#	Practice	Category	References
13	DevOps team must provide safe deployment parameters to avoid excessive workload on infrastructure	Deployment	[15]
14	Production code is pair programmed and collectively owned	Development	[42]
15	Use trunk-based development over long-lived feature branches	Development	[9, 33]
16	Use Test driven Development where all code has unit tests	Development	[42]
17	Keep variance (code/quality/behaviour) between development and production to minimum	Development	[15, 34]
18	Encourage full collaboration and transparency between developers and operators	DevOps Team	[4, 15, 76]
19	DevOps team's needs are clarified and manual processes are identified	DevOps Team	[34]
20	Provide a real-time and continuous collaboration system	DevOps Team	[15]
21	DevOps team synchronizes critical services	Knowledge Sharing	[15]
22	DevOps team must have clear insight into the SD project	Knowledge Sharing	[15]
23	DevOps team must be able to increase release frequency to satisfy business demand	Lead Time	[15]
24	Support configurable logging that can optionally be turned on/off as needed to debug issues	Logging	[32]
25	Automated and continuous monitoring of applications and resources	Monitoring	[4, 9, 22, 32]
26	Automated dashboards that include health checks and performance (throughput, and latency)	Monitoring	[4, 32]
27	Teams can update systems based on monitoring reports or error logs	Monitoring	[15, 32]

Continued on next page

Table B.1 – continued from previous page

#	Practice	Category	References
28	Risks are monitored through acceptance testing	Monitoring	[42]
29	Integrated and continuous planning	Planning	[4]
30	Production support includes required utilities	Production	[4, 32]
31	Use Agile and LEAN practices (e.g. sprint planning and requirements engineering)	Project Management	[4, 9, 33, 34]
32	Applications are prototyped	Prototyping	[4]
33	Code reviews are change based	Quality Assurance	[9]
34	Perform Retro-Quality Assurance tests in the build pipeline	Quality Assurance	[15]
35	Make small and continuous releases with feedback loops	Release	[32, 41, 42]
36	Use single-version software	Resources	[32]
37	DevOps team must provide self-service and resources management of platform	Resources	[15]
38	Create development sandboxes for minimum code deployment	Sandboxes	[15]
39	Collaborate with stakeholders often and provide overall visibility into the project	Stakeholders	[4, 15, 42]
40	Processes are standardized	Standardisation	[4]
41	Automated and continuous testing in development and staging environments	Testing	[4, 15, 32, 33, 42]
42	Enforce effective test data management	Testing	[33]
43	Reduce the time it takes to test, validate and QA code	Testing	[34]
44	Applications are modelled and simulated	Testing	[4]
45	Use DevOps tools to improve and optimise work	Tooling	[15, 34]
46	Allow teams to choose their own DevOps tools	Tooling	[76]
47	Consider the cultural change from both the perspective of the driver and the participants	Transform	[76]

Appendix C

Tools Found in Literature

Table C.1 displays all tools identified in scientific literature. Please note that tools can fall under multiple categories.

Table C.1: *Tools identified in scientific literature.*

#	Tool	Category	References
1	Apache Ant	Build	[13, 35, 44]
2	Maven	Build	[13, 34, 34, 35, 44]
3	Rake	Build	[13]
4	Gradle	Build	[13, 34, 35, 44]
5	AWS	Cloud Computing	[9, 11, 35]
6	Heroku	Cloud Computing	[15, 34]
7	OpenStack	Cloud Computing	[34, 45]
8	Azure	Cloud Computing	[34]
9	Google Cloud	Cloud Computing	[34]
10	Rackspace	Cloud Computing	[34]
11	SonarQube	Code Analyser	[9, 35, 46]
12	Coverity	Code Analyser	[35]
13	Fortify	Code Analyser	[35]
14	JS hint	Code Analyser	[35]
15	Blue optima	Code Analyser	[35]
16	CheckStyle	Code Analyser	[34, 35]
17	Gerrit	Code Analyser	[35]
18	Jacoco	Code Analyser	[35]
19	Clover	Code Analyser	[35]
20	Semmle	Code Analyser	[35]
21	SonarLint	Code Analyser	[34, 46]
22	FindBugs	Code Analyser	[34]
23	Slack	Collaboration	[15, 34]
24	HipChat	Collaboration	[15]
25	Puppet	Configuration Management	[6, 9, 13, 34, 35, 44, 47]
26	Chef	Configuration Management	[9, 13, 34, 35, 44, 47, 48]
27	Ansible	Configuration Management	[9, 13, 34, 35, 44]
28	SaltStack	Configuration Management	[34, 35]
29	CFengine	Configuration Management	[35, 44]

Continued on next page

Table C.1 – continued from previous page

#	Tool	Category	References
30	Windows Server IIS7	Configuration Management	[35]
31	RANCID	Configuration Management	[44]
32	Hiera	Configuration Management	[6]
33	UrbanCode	Continuous Delivery	[47]
34	Jenkins	Continuous Integration	[9, 13, 35, 44, 46, 47]
35	Bamboo	Continuous Integration	[13, 35, 49]
36	ThoughtWorks' GoCD	Continuous Integration	[1, 11]
37	Rundeck	Continuous Integration	[35]
38	CircleCI	Continuous Integration	[35]
39	CruiseControl	Continuous Integration	[44]
40	TeamCity	Continuous Integration	[6, 11, 13, 35]
41	Codeship	Continuous Integration	[15]
42	Travis CI	Continuous Integration	[15, 35]
43	MongoDB	Database Management	[15]
44	DBMeastro	Database Management	[34]
45	Liquibase	Database Management	[6, 34]
46	Datical	Database Management	[34]
47	HockeyApp	Delivery Server	[49]
48	Octopus Deploy	Deployment	[11, 34]
49	uDeploy	Deployment	[35, 47]
50	Juju	Deployment	[48]
51	Capistrano	Deployment	[34]
52	CodeDeploy	Deployment	[34]
53	Wiki	Documentation	[12]
54	Cloud Hosting	Hosting	[11]
55	Terraform	Infrastructure as Code	[11]
56	Confluence	Knowledge Sharing	[34]
57	Loggly	Logging	[13, 15, 44]
58	Greylog	Logging	[9, 13]
59	Papertrail	Logging	[15, 44]
60	Logstash	Logging	[34, 35, 44]
61	Kibina	Logging	[44]
62	Sumo Logic	Logging	[44]
63	Log.io	Logging	[34]
64	Nagios	Monitoring	[13, 15, 34, 35, 44]
65	New Relic	Monitoring	[9, 13, 15, 34]
66	Cacti	Monitoring	[13, 44]
67	MONIT	Monitoring	[35]
68	Splunk	Monitoring	[34, 35, 44]
69	AppPerfect	Monitoring	[35]
70	Ganglia	Monitoring	[44]
71	Graphite	Monitoring	[44]
72	Sensu	Monitoring	[44]
73	Zabbix	Monitoring	[34, 44]
74	Kineses	Process Analysis	[9]
75	Jira	Project Management	[12, 35, 46, 49]
76	Linux Bash	Scripting	[34]

Continued on next page

Table C.1 – continued from previous page

#	Tool	Category	References
77	Windows Powershell	Scripting	[34]
78	Vault	Security	[34]
79	Yum	Software Package Manager	[6]
80	SPACE4Cloud	System Performance	[50]
81	Cucumber	Testing	[11, 15]
82	Selenium	Testing	[9, 11, 34, 35]
83	Junit	Testing	[15, 34, 35]
84	TestNG	Testing	[34]
85	Jmeter	Testing	[34]
86	Dynatrace	Testing	[34]
87	GitHub	Version Control Management	[6, 9, 11, 15, 35]
88	Bitbucket	Version Control Management	[9, 15]
89	GitLab	Version Control Management	[35, 46]
90	Stash	Version Control Management	[49]
91	Nexus	Version Control Management	[34, 35]
92	Archiva	Version Control Management	[35]
93	SharePoint	Version Control Management	[35]
94	Artifactory	Version Control Management	[35]
95	Deveo	Version Control Management	[9]
96	Helix Core	Version Control Management	[35]
97	SVN	Version Control Management	[34]
98	Docker	Virtualisation	[6, 9, 34, 35, 45, 51]
99	Vagrant	Virtualisation	[34]

Appendix D

DevOps & Agile Maturity by Industry

Figures D.2 and D.1 show the DevOps and Agile maturity per industry, respectively.

Figure D.1: *DevOps maturity per industry.*

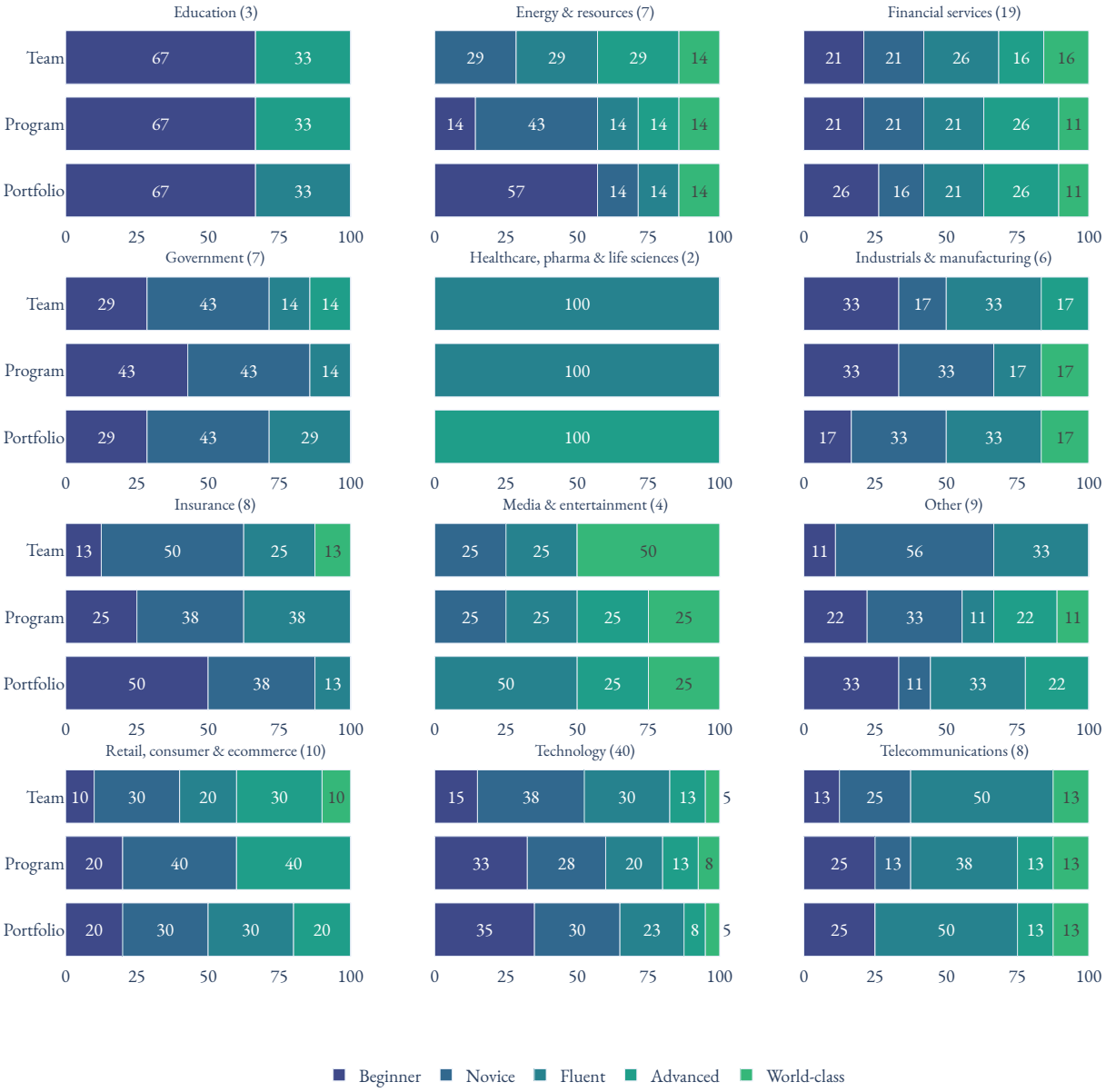


Figure D.2: *Agile maturity per industry.*

Appendix E

Survey Questions

DevOps Survey

Start of Block: Intro

Q1
Dear participant,

Thank you for taking the time to participate in this anonymous survey. The research around this survey investigates DevOps technologies and practices. We are interested if certain tools and/or technologies have effects on the performance of organizations.

Benefits of filling in the survey Gain insights into the tools/practices that other organizations are using Gain insights into how specific tools/practices impact organizational performance Gain insights into how the maturity of your organization compares to others in the industry
This survey is meant for people who are working or have worked with DevOps. This survey is fully anonymous and all data will be treated fully confidential. It is estimated that the survey will take around 15-20 minutes to answer.

Thank you for your participation.

Kind regards,

Robert Blinde
s1370162@vuw.leidenuniv.nl
Master student ICT in Business at Leiden University

Dr. C.J. Stettina MSc
c.j.stettina@liacs.leidenuniv.nl Leiden University

T.D. Offerman MSc
t.d.offerman@liacs.leidenuniv.nl
Leiden University

End of Block: Intro

Start of Block: Demographics / Organization

Q2
What is the industry you primarily work in?

- ○ Education (16)
- ○ Energy & resources (17)
- ○ Financial services (18)
- ○ Government (19)
- ○ Healthcare, pharma & life sciences (20)
- ○ Industrials & manufacturing (21)
- ○ Insurance (22)
- ○ Media & entertainment (23)
- ○ Nonprofit (24)
- ○ Retail, consumer & ecommerce (25)
- ○ Technology (26)
- ○ Telecommunications (27)
- ○ Other (28)

Page Break

Q3 What is your role within your organization?
Please choose the role that best describes you.

- ○ Automation Engineer or Automation Expert (107)
- ○ Information Security Operator (108)
- ○ IT Operations Engineer or Infrastructure Engineer (109)
- ○ Network Operations Engineer (110)
- ○ Quality Engineer or Assurance Professional (111)
- ○ Security Engineer (112)
- ○ Software Developer or Software Engineer (113)
- ○ Tester / Software Analyst (114)
- ○ User Experience Designer (115)
- ○ Project Manager (102)
- ○ Product Manager (103)
- ○ Release Manager (104)
- ○ Sponsor (98)
- ○ Agile Coach (99)
- ○ DevOps Coach (100)
- ○ Other consultant (105)
- ○ Other *(please specify)*: (116) _____

Page Break

Q4
How many employees work at your organization?

- ○ 1 - 4 (9)
- ○ 5 - 9 (10)
- ○ 10 - 19 (11)
- ○ 20 - 99 (12)
- ○ 100 - 499 (13)
- ○ 500 - 1,999 (14)
- ○ 2,000 - 4,999 (15)
- ○ 5,000 - 9,999 (16)
- ○ 10,000 + (17)
- ○ Don't know / NA (18)

Page Break

Q5 How many years of experience with DevOps do you have?

- ○ 0 (6)
- ○ 1 - 2 (7)
- ○ 3 - 5 (8)
- ○ 6 - 10 (9)
- ○ 11 - 15 (10)
- ○ > 16 (11)

End of Block: Demographics / Organization

Start of Block: Transformation

Q6 Is your organization currently undergoing or has it undergone an agile or DevOps transformation?

- ○ Yes, my organization is currently undergoing a transformation (1)
- ○ Yes, my organization completed a transformation (2)
- ○ No, but my organization is about to start a transformation (3)
- ○ No, not at all (4)

Skip To: End of Block If Is your organization currently undergoing or has it undergone an agile or DevOps transformation? = No, not at all

Page Break

Q7
What is the (expected) duration of the transformation?

- ○ 6 months or less (1)
- ○ Between 6 and 12 months (2)
- ○ Between 1 and 2 years (3)
- ○ Between 2 and 5 years (4)
- ○ More than 5 years (5)

Page Break

Q8 Which large scale framework is used during the transformation?

- [ ] Agile Portfolio Management (APM) (27)
- [ ] Disciplined Agile Delivery (DAD) (29)
- [ ] Enterprise scrum (31)
- [ ] Internally created methods (32)
- [ ] Large scale scrum (LESS) (33)
- [ ] Lean Management (34)
- [ ] Nexus (35)
- [ ] Scaled Agile Framework (SAFe) (36)
- [ ] Spotify Model (43)
- [ ] Scrum of scrums (37)
- [ ] Waterfall (42)
- [ ] None of the above (40)
- [ ] Other *(please specify)*: (41) _____

Page Break

Q9 Which DevOps framework is used during the transition?

- [ ] Culture, Automation, Lean, Measurement, and Sharing (CALMS) (1)
- [ ] SAFe's CALMR (4)
- [ ] CAMS (5)
- [ ] SQUID Architecture Framework (8)
- [ ] None of the above (3)
- [ ] Other *(please specify):* (6) _____

End of Block: Transformation

Start of Block: Maturity Models

Q10
Below you find five levels of the *Portfolio* aspect from an Agile Maturity Model.

Where in this model do you consider your organization currently?

- ○ Beginner (6)
- ○ Novice (7)
- ○ Fluent (8)
- ○ Advanced (9)
- ○ World-class (10)

Page Break

Q11
Below you find five levels of the *Program* aspect from an Agile Maturity Model.

Where in this model do you consider your organization currently?

- ○ Beginner (6)
- ○ Novice (7)
- ○ Fluent (8)
- ○ Advanced (9)
- ○ World-class (10)

Page Break

Q12
Below you find five levels of the *Team* aspect from an Agile Maturity Model.

Where in this model do you consider your organization currently?

- ○ Beginner (6)
- ○ Novice (7)
- ○ Fluent (8)
- ○ Advanced (9)
- ○ World-class (10)

Page Break

Q13
Below you find four levels of the *Organization & Culture* aspect from a DevOps Maturity Model.

Where in this model do you consider your organization currently?

- ○ Level 1 (13)
- ○ Level 2 (14)
- ○ Level 3 (15)
- ○ Level 4 (16)

Page Break

Q14
Below you find four levels of the *Environments & Release* aspect from a DevOps Maturity Model.

Where in this model do you consider your organization currently?

- ○ Level 1 (13)
- ○ Level 2 (14)
- ○ Level 3 (15)
- ○ Level 4 (16)

Page Break

Q15
Below you find four levels of the *Builds & Continuous Release* aspect from a DevOps Maturity Model.

Where in this model do you consider your organization currently?

- ○ Level 1 (13)
- ○ Level 2 (14)
- ○ Level 3 (15)
- ○ Level 4 (16)

Page Break

Q16
Below you find four levels of the *Quality Assurance* aspect from a DevOps Maturity Model.

Where in this model do you consider your organization currently?

- ○ Level 1 (13)
- ○ Level 2 (14)
- ○ Level 3 (15)
- ○ Level 4 (16)

Page Break

Q17
Below you find four levels of the *Visibility & Reporting* aspect from a DevOps Maturity Model.

Where in this model do you consider your organization currently?

- ○ Level 1 (13)
- ○ Level 2 (14)
- ○ Level 3 (15)
- ○ Level 4 (16)

Page Break

Q18
Below you find four levels of the *Technology & Architecture* aspect from a DevOps Maturity Model.

Where in this model do you consider your organization currently?

- ○ Level 1 (13)
- ○ Level 2 (14)
- ○ Level 3 (15)
- ○ Level 4 (16)

End of Block: Maturity Models

Start of Block: Tools & Practices

Q19 Where are you currently hosting your, through DevOps developed, software?

- ○ A public cloud (1)
- ○ Multiple public clouds (2)
- ○ A private cloud (3)
- ○ Hybrid cloud (combination of public and private clouds) (4)
- ○ On-premise datacenter (5)
- ○ Other *(please specify)*: (6) _____

Page Break

Q20 How are you currently deploying your, through DevOps developed, software?

- ○ Containers (Docker, Kubernetes) (1)
- ○ FaaS (AWS Lambda, Google Cloud Functions) (2)
- ○ PaaS (Heroku, App Engine, Elastic Beanstalk) (3)
- ○ Virtual Machines (VM's) (5)
- ○ Other *(please specify)*: (6) _____

Page Break

Q21 How much of your time do you spend working on the following?
_____ Features (1)
_____ Enablers (2)
_____ Technical debt (3)
_____ Maintenaince (22)

Page Break

Q22 What percentage of your time do you spend on creating business features vs. improving infrastructure?

_____ Business features (1)

_____ Improving infrastructure (2)

Page Break

Q23 What percentage of your time do you spent on unplanned work?

Not Applicable

0 10 20 30 40 50 60 70 80 90 100

% unplanned work ()	

Page Break

Q24
How quickly can developers see the results of integration tests?

- Less than 1 minute (25)
- 1 minute - 10 minutes (26)
- 10 minutes - 1 hour (27)
- 1 hour - 1 day (28)
- 1 day - 1 week (29)
- 1 week - 1 month (30)
- More than 1 month (31)
- We do not have any testing in place (32)

Page Break

Q25 The automated DevOps pipeline includes the following:
Select all that apply.

- [ ] Integration with production monitoring and observability tools (7)
- [ ] Integration with chatbots/Slack (8)
- [ ] Automated provisioning and deployment to testing environments (9)
- [ ] Automated performance tests (10)
- [ ] Automated security tests (11)
- [ ] Automated unit tests (12)
- [ ] Automated acceptance tests (13)
- [ ] Automated build (14)
- [ ] Automated deployment to production (15)
- [ ] None of the above (16)
- [ ] Other *(please specify)*: (17) _____
- [ ] Other *(please specify)*: (18) _____
- [ ] Other *(please specify)*: (19) _____

Page Break

Q26 Do you make extensive use of open source components, libraries, and platforms?

- ○ Strongly agree (9)
- ○ Somewhat agree (10)
- ○ Neither agree nor disagree (11)
- ○ Somewhat disagree (12)
- ○ Strongly disagree (13)
- ○ Don't know / NA (14)

Page Break

Q27
How do you measure customer satisfaction?

_____

_____

_____

_____

_____

End of Block: Tools & Practices

Start of Block: T&P: CI + CD

Q28
The next questions will be about the practices, patterns, and technologies you use at work. Please select how often you adhere to each practice.

Q29 For your organization's DevOps pipeline, how often are automated and continuous deployments part of it?

- Always (9)
- Most of the time (10)
- About half the time (11)
- Sometimes (12)
- Never (13)

Page Break

Q30 Does your organization make small and continuous releases?

- ○ Always (29)
- ○ Most of the time (30)
- ○ About half the time (31)
- ○ Sometimes (32)
- ○ Never (33)

Page Break

Q31 Do developers get feedback based on releases?

- ○ Always (9)
- ○ Most of the time (10)
- ○ About half the time (11)
- ○ Sometimes (12)
- ○ Never (13)

Page Break

Q32 Are sandboxes created for minimum code deployment?

- ○ Always (9)
- ○ Most of the time (10)
- ○ About half the time (11)
- ○ Sometimes (12)
- ○ Never (13)

Page Break

Q33 Which tools do you use for automated and continuous **delivery**?
Select all that apply.

- [ ] Bamboo (4)
- [ ] CircleCI (14)
- [ ] Codeship (13)
- [ ] Jenkins (6)
- [ ] TeamCity (7)
- [ ] ThoughtWorks' GoCD (15)
- [ ] Travis CI (8)
- [ ] Visual Studio App Center (12)
- [ ] None of the above (17)
- [ ] Other *(please specify)*: (9) _____
- [ ] Other *(please specify)*: (10) _____
- [ ] Other *(please specify)*: (11) _____

Page Break

Q34 Which tools do you use for automated and continuous **deployment**?
Select all that apply.

- [ ] Bamboo (57)
- [ ] Capistrano (66)
- [ ] CodeDeploy (58)
- [ ] Juju (59)
- [ ] Octopus Deploy (60)
- [ ] Travis CI (65)
- [ ] uDeploy (61)
- [ ] None of the above (68)
- [ ] Other *(please specify)*: (62) _____
- [ ] Other *(please specify)*: (63) _____
- [ ] Other *(please specify)*: (64) _____

Page Break

Q35 Which tools do you use for automated and continuous **integration**?
Select all that apply.

- ☐ Bamboo (4)
- ☐ CircleCI (5)
- ☐ Codeship (6)
- ☐ CruiseControl (7)
- ☐ Jenkins (8)
- ☐ Rundeck (9)
- ☐ TeamCity (10)
- ☐ ThoughtWorks' GoCD (11)
- ☐ Travis CI (12)
- ☐ None of the above (16)
- ☐ Other *(please specify)*: (13) _____
- ☐ Other *(please specify)*: (14) _____
- ☐ Other *(please specify)*: (15) _____

End of Block: T&P: CI + CD

Start of Block: T&P: Configuration

Q36 Is everything stored as code and under version control?
(Everything includes application code, infrastructure, deployment configuration, etc.)

- ○ Strongly agree (38)
- ○ Somewhat agree (39)
- ○ Neither agree nor disagree (40)
- ○ Somewhat disagree (41)
- ○ Strongly disagree (42)

Page Break

Q37 Does your organization use configuration management?

- ○ Always (9)
- ○ Most of the time (10)
- ○ About half the time (11)
- ○ Sometimes (12)
- ○ Never (13)

Page Break

Display This Question:

If Is everything stored as code and under version control?(Everything includes application code, inf... != Strongly disagree

Q38 Which tools do you use for version control management?
Select all that apply.

- [ ] Archiva (4)
- [ ] Artifactory (5)
- [ ] Bitbucket (6)
- [ ] Deveo (7)
- [ ] GitHub (8)
- [ ] GitLab (9)
- [ ] Helix Core (10)
- [ ] Nexus (11)
- [ ] SharePoint (12)
- [ ] Stash (13)
- [ ] SVN (14)
- [ ] None of the above (18)
- [ ] Other *(please specify)*: (15) _____
- [ ] Other *(please specify)*: (16) _____
- [ ] Other *(please specify)*: (17) _____

Page Break

Display This Question:

If Does your organization use configuration management? != Never

Q39 Which tools do you use for configuration management?
Select all that apply.

- ☐ Ansible (29)
- ☐ CFengine (30)
- ☐ Chef (31)
- ☐ Puppet (32)
- ☐ RANCID (33)
- ☐ SaltStack (34)
- ☐ Windows Server IIS7 (35)
- ☐ None of the above (39)
- ☐ Other *(please specify)*: (36) _____
- ☐ Other *(please specify)*: (37) _____
- ☐ Other *(please specify)*: (38) _____

End of Block: T&P: Configuration

Start of Block: T&P: Testing

Q40 Is there automated and continuous testing in development and staging environments?

- ○ Always (19)
- ○ Most of the time (20)
- ○ About half the time (21)
- ○ Sometimes (22)
- ○ Never (23)

Page Break

Q41 Do you actively try to reduce the time it takes to test, validate and QA code?

- ○ Strongly agree (14)
- ○ Somewhat agree (15)
- ○ Neither agree nor disagree (16)
- ○ Somewhat disagree (17)
- ○ Strongly disagree (18)

Page Break

Q42 Are code reviews change-based?

- ○ Always (9)
- ○ Most of the time (10)
- ○ About half the time (11)
- ○ Sometimes (12)
- ○ Never (13)

Page Break

Display This Question:

If Is there automated and continuous testing in development and staging environments? != Never

Q43 Which tools do you use for testing?
Select all that apply.

- ☐ Cucumber (50)
- ☐ Dynatrace (51)
- ☐ Jmeter (52)
- ☐ Junit (53)
- ☐ Selenium (54)
- ☐ TestNG (55)
- ☐ None of the above (59)
- ☐ Other *(please specify)*: (56) _____
- ☐ Other *(please specify)*: (57) _____
- ☐ Other *(please specify)*: (58) _____

Page Break

Q44 Which code analyzer tools do you use?
Select all that apply.

- [ ] Blue optima (4)
- [ ] CheckStyle (5)
- [ ] Clover (6)
- [ ] Coverity (7)
- [ ] FindBugs (8)
- [ ] Fortify (9)
- [ ] Gerrit (10)
- [ ] Jacoco (11)
- [ ] JS hint (12)
- [ ] Semmle (13)
- [ ] SonarLint (14)
- [ ] SonarQube (15)
- [ ] None of the above (19)
- [ ] Other *(please specify)*: (16) _____
- [ ] Other *(please specify)*: (17) _____

☐ Other *(please specify)*: (18) _____

End of Block: T&P: Testing

Start of Block: T&P: Monitoring

Q45
Are applications and resources automatically and continuously monitored?

- ○ Always (9)
- ○ Most of the time (10)
- ○ About half the time (11)
- ○ Sometimes (12)
- ○ Never (13)

Page Break

Q46 Are automated dashboards that include health checks and performance (e.g. throughput, and latency) in place?

- ○ Strongly agree (14)
- ○ Somewhat agree (15)
- ○ Neither agree nor disagree (16)
- ○ Somewhat disagree (17)
- ○ Strongly disagree (18)

Page Break

Display This Question:

If Are applications and resources automatically and continuously monitored? != Never

Or Are automated dashboards that include health checks and performance (e.g. throughput, and latency... != Strongly disagree

Q47 Which tools do you use for automated and continuous monitoring?
Select all that apply.

- [ ] AppPerfect (14)
- [ ] Cacti (15)
- [ ] Ganglia (16)
- [ ] Graphite (17)
- [ ] MONIT (18)
- [ ] Nagios (19)
- [ ] New Relic (20)
- [ ] Sensu (21)
- [ ] Splunk (22)
- [ ] Zabbix (23)
- [ ] None of the above (27)
- [ ] Other *(please specify)*: (24) _____
- [ ] Other *(please specify)*: (25) _____
- [ ] Other *(please specify)*: (26) _____

End of Block: T&P: Monitoring

Start of Block: T&P: Logging

Q48 Do you support logging that can be turned on/off via configuration?
(Configuration can be in the form of files, env. variables, etc.)

- ○ Always (9)
- ○ Most of the time (10)
- ○ About half the time (11)
- ○ Sometimes (12)
- ○ Never (13)

Page Break

Display This Question:

If Do you support logging that can be turned on/off via configuration?(Configuration can be in the f... != Never

Q49
Which tools do you use for logging?

- [ ] Greylog (11)
- [ ] Kibina (12)
- [ ] Log.io (13)
- [ ] Loggly (14)
- [ ] Logstash (15)
- [ ] Papertrail (16)
- [ ] Sumo Logic (17)
- [ ] None of the above (21)
- [ ] Other *(please specify)*: (18) _____
- [ ] Other *(please specify)*: (19) _____
- [ ] Other *(please specify)*: (20) _____

End of Block: T&P: Logging

Start of Block: T&P: Development

Q50 Is trunk-based development used over long-lived feature branches?

- Always (9)
- Most of the time (10)
- About half the time (11)
- Sometimes (12)
- Never (13)

Page Break

Q51 How often is test-driven development used where all code has unit tests used?

- Always (9)
- Most of the time (10)
- About half the time (11)
- Sometimes (12)
- Never (13)

End of Block: T&P: Development

Start of Block: Impact of DevOps

Q52 How often does your organization deploy code for its primary service or application?

- on demand (multiple deploys per day) (1)
- between once per hour and once per day (2)
- between once per day and once per week (3)
- between once per week and once per month (4)
- between once per month and once every six months (5)
- fewer than once every six months (6)
- Don't know / NA (7)

Page Break

Q53 How much time does it take to go from code committed to code successfully running in production?

- ○ less than one hour (1)
- ○ less than one day (2)
- ○ between one day and one week (3)
- ○ between one week and one month (4)
- ○ between one month and six months (5)
- ○ more than six months (6)
- ○ Don't know / NA (8)

Page Break

Q54
How long does it generally take your organization to restore service for a service/application when a service incident occurs?
(restoration of service includes: hotfixes, rollbacks, patches, fix-forwards, etc.)

- ○ less than one hour (4)
- ○ less than one day (5)
- ○ between one day and one week (6)
- ○ between one week and one month (7)
- ○ between one month and six months (8)
- ○ more than six months (9)
- ○ Don't know / NA (10)

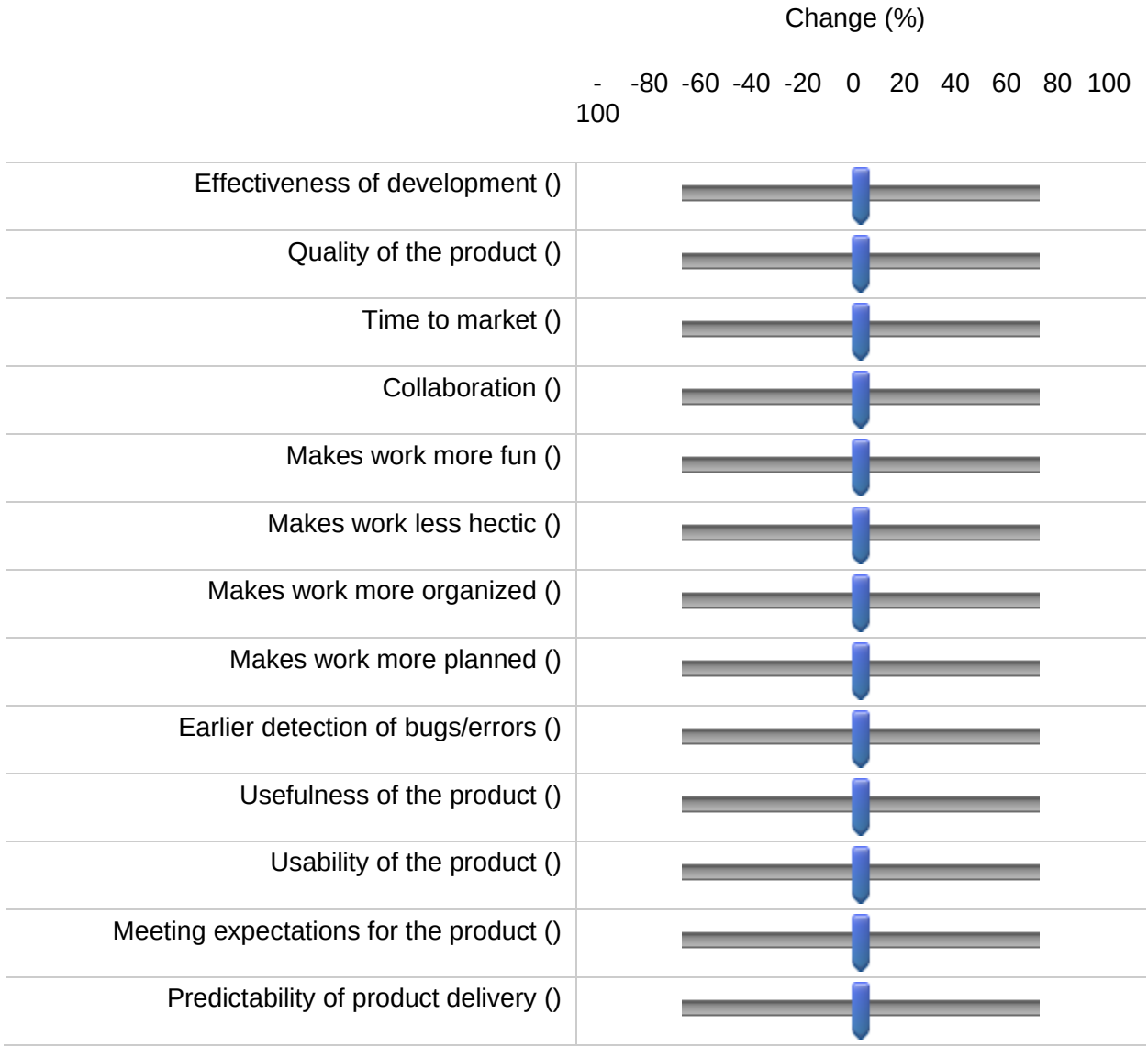
Page Break

Q55 What percentage of changes for the primary service or application results in degraded service or requires remediation?

- ○ Less than 1% (11)
- ○ 1% - 5% (12)
- ○ 6% - 15% (13)
- ○ 16% - 30% (14)
- ○ 31% - 45% (15)
- ○ 46% - 60% (16)
- ○ 61% - 75% (17)
- ○ 76% - 100% (18)
- ○ Don't know / NA (19)

Page Break

Q56 How does the implementation of DevOps in your organization impact (positively or negatively) the following topics?



End of Block: Impact of DevOps

Start of Block: Firm Performance

Q57 Over the past year, how well did your organization meet its goals regarding **overall organizational performance**?

- ○ Performed well above goals (4)
- ○ Performed above goals (5)
- ○ Performed slightly above goals (6)
- ○ Met goals (7)
- ○ Performed slightly below goals (8)
- ○ Performed below goals (9)
- ○ Performed well below goals (10)
- ○ Don't know / NA (11)

Page Break

Q58 Over the past year, how well did your organization meet its goals regarding **overall organizational profitability**?

- ○ Performed well above goals (4)
- ○ Performed above goals (5)
- ○ Performed slightly above goals (6)
- ○ Met goals (7)
- ○ Performed slightly below goals (8)
- ○ Performed below goals (9)
- ○ Performed well below goals (10)
- ○ Don't know / NA (11)

Page Break

Q59 Over the past year, how well did your organization meet its goals regarding **customer satisfaction**?

- ○ Performed well above goals (4)
- ○ Performed above goals (5)
- ○ Performed slightly above goals (6)
- ○ Met goals (7)
- ○ Performed slightly below goals (8)
- ○ Performed below goals (9)
- ○ Performed well below goals (10)
- ○ Don't know / NA (11)

Page Break

Q60 Over the past year, how well did your organization meet its goals regarding **quality of products and services**?

- Performed well above goals (4)
- Performed above goals (5)
- Performed slightly above goals (6)
- Met goals (7)
- Performed slightly below goals (8)
- Performed below goals (9)
- Performed well below goals (10)
- Don't know / NA (11)

Page Break

Q61 Over the past year, how well did your organization meet its goals regarding **operating efficiency**?

- Performed well above goals (1)
- Performed above goals (4)
- Performed slightly above goals (5)
- Met goals (6)
- Performed slightly below goals (7)
- Performed below goals (8)
- Performed well below goals (9)
- Don't know / NA (10)

Page Break

Q62 Over the past year, how well did your organization meet its goals regarding **achieving the organizational and mission goals**?

- Performed well above goals (1)
- Performed above goals (4)
- Performed slightly above goals (5)
- Met goals (6)
- Performed slightly below goals (7)
- Performed below goals (8)
- Performed well below goals (9)
- Don't know / NA (10)

End of Block: Firm Performance

Appendix F

Time Allocation & Maturity

Figures F.1 and F.2 show time allocation across Agile and DevOps layers, respectively.

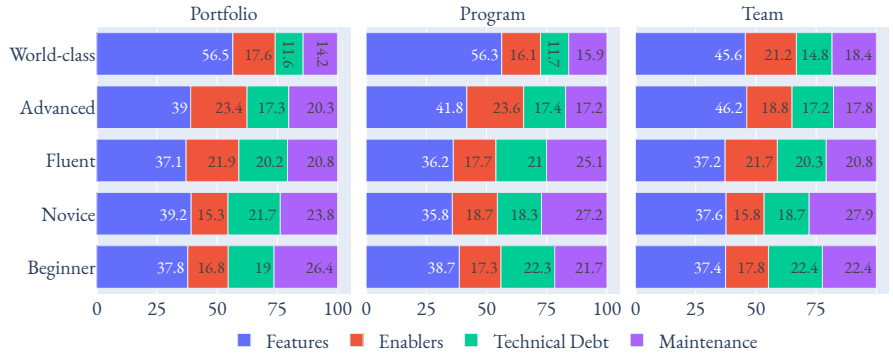


Figure F.1: *Time allocation across three Agile layers.*

Figure F.2: *Time allocation across six DevOps layers.*

Appendix G

Tools Used by Survey Participants

The following tables show the DevOps tools used by the survey participants.

Table G.1: *Continuous delivery tools used by participants.*

#	Tool	Usage
1	Jenkins	37
2	Azure DevOps	9
3	GitLab CI	9
4	Bamboo	5
5	Jenkins/Visual Studio App Center	4
6	GitLab CI/Jenkins	3
7	TeamCity	3
8	Azure DevOps/Jenkins	3
9	Bamboo/Jenkins	3
10	Bitbucket Pipelines	2
11	Visual Studio App Center	2
12	Jenkins/XL Deploy	2
13	Octopus Deploy	2
14	Azure DevOps/Visual Studio App Center	1
15	Azure DevOps/GitHub Actions	1
16	GitLab CI/TeamCity	1
17	GitHub Actions/GitLab CI	1
18	CircleCI/TeamCity/Travis CI/Visual Studio App Center	1
19	Azure DevOps/GitHub Actions/Jenkins/Travis CI	1
20	Spinnaker	1
21	Codeship	1
22	GitHub Actions	1
23	Bitbucket Pipelines/Jenkins	1
24	Azure DevOps/Jenkins/Visual Studio App Center	1
25	Buildkite	1
26	Azure DevOps/TeamCity	1
27	Bamboo/Jenkins/Travis CI	1
28	Azure DevOps/Jenkins/XL Release	1
29	Drone CI	1

Continued on next page

Table G.1 – continued from previous page

#	Tool	Usage
30	CircleCI/Jenkins	1
31	OutSystems/Visual Studio App Center	1
32	Ansible/GitLab CI	1
33	AWS ECS/GitHub Actions/Serverless	1
34	Bitrise/GitLab CI	1
35	Jenkins/XL Deploy/XL Release	1
36	Consul/Jenkins/Nomad	1
37	Bamboo/CircleCI/Jenkins/Visual Studio App Center	1
38	Codefresh/GitHub Actions	1
39	Semaphore CI	1
40	Bamboo/TeamCity	1

Table G.2: *Continuous deployment tools used by participants.*

#	Tool	Usage
1	Azure DevOps	14
2	GitLab CI	10
3	CodeDeploy	10
4	Octopus Deploy	5
5	Bamboo	4
6	Jenkins	4
7	Travis CI	3
8	Bitbucket Pipelines	2
9	Juju	2
10	XL Deploy	2
11	Ansible	2
12	Azure DevOps/GitHub Actions	1
13	GitHub Actions/GitLab CI/Terraform	1
14	Capistrano/CodeDeploy/Octopus Deploy/Travis CI	1
15	GitLab CI/Jenkins	1
16	Cloudbuild/Spinnaker	1
17	Python script/Shell script	1
18	Bitbucket Pipelines/CodeDeploy	1
19	Shell script	1
20	Buildkite/Shell script	1
21	Ansible/Terraform/XL Deploy/uDeploy	1
22	Ansible/GitLab CI	1
23	Bamboo/Capistrano	1
24	Azure DevOps/Jenkins	1
25	Flux/Kustomization	1
26	AWS ECS/GitHub Actions/Serverless	1
27	Nomad	1
28	Bamboo/CodeDeploy/Travis CI	1
29	GitLab CI/Terraform	1
30	Codefresh/GitHub Actions	1
31	Bamboo/CodeDeploy	1

Table G.3: *Continuous integration tools used by participants.*

#	Tool	Usage
1	Jenkins	41
2	Azure DevOps	15
3	GitLab CI	11
4	Bamboo	5
5	Bitbucket Pipelines	3
6	GitLab CI/Jenkins	3
7	Azure DevOps/Jenkins	3
8	Codeship	2
9	TeamCity	2
10	CircleCI	2
11	Azure DevOps/GitHub Actions	1
12	CircleCI/Codeship/CruiseControl/Jenkins/TeamCity/GoCD	1
13	Azure DevOps/GitHub Actions/Jenkins/Travis CI	1
14	CruiseControl	1
15	Buildkite/Kubernetes/Terraform	1
16	Rundeck/TeamCity	1
17	Drone CI	1
18	Flux	1
19	Codeship/Jenkins	1
20	GitHub Actions	1
21	Bamboo/CircleCI	1
22	Bitrise/GitLab CI	1
23	GitLab CI/Jenkins/SVN	1
24	Bamboo/CircleCI/Jenkins/Rundeck/Travis CI	1
25	Codefresh/GitHub Actions	1
26	Semaphore CI	1
27	Cucumber/TeamCity	1
28	Bamboo/Jenkins	1

Table G.4: *Version control management tools used by participants.*

#	Tool	Usage
1	GitHub	23
2	Bitbucket	15
3	Azure DevOps	13
4	GitLab	11
5	GitHub/GitLab	6
6	Bitbucket/GitHub	4
7	GitLab/Nexus	3
8	Bitbucket/GitHub/Nexus	2
9	Azure DevOps/GitHub	2
10	OutSystems	2

Continued on next page

Table G.4 – continued from previous page

#	Tool	Usage
11	Artifactory/GitHub	2
12	Bitbucket/GitLab	1
13	Artifactory/GitLab	1
14	SharePoint	1
15	Archiva/Bitbucket/GitHub/Helix Core/Nexus/SVN/Stash	1
16	Azure DevOps/Bitbucket/GitHub/Nexus	1
17	CVS/GitLab/SVN	1
18	Azure DevOps/Bitbucket	1
19	GitHub/GitLab/SVN	1
20	PTC	1
21	GitHub/SharePoint	1
22	Git	1
23	Artifactory/Azure DevOps/Bitbucket/Nexus	1
24	GitHub/Stash	1
25	Azure DevOps/Bitbucket/Nexus/SVN	1
26	Artifactory/Bitbucket/GitHub/GitLab	1
27	Bitbucket/GitHub/GitLab/Stash	1
28	Bitbucket/GitHub/GitLab	1
29	Artifactory/Azure DevOps/Stash	1
30	Archiva/GitHub	1
31	SVN	1
32	Git/LifeTime	1
33	Bitbucket/Nexus	1
34	Artifactory/Bitbucket	1
35	Artifactory/GitLab/SVN	1
36	Artifactory/Bitbucket/GitHub/Nexus	1
37	Deveo	1
38	Azure DevOps/Git	1
39	Bitbucket/Nexus/SharePoint	1
40	Artifactory	1
41	Azure DevOps/GitLab/SharePoint	1
42	GitHub/GitLab/SharePoint	1
43	Bitbucket/GitLab/SVN	1
44	Artifactory/Bitbucket/GitHub	1

Table G.5: *Configuration management tools used by participants.*

#	Tool	Usage
1	Ansible	36
2	Ansible/Puppet	6
3	Puppet	6
4	Terraform	4
5	Chef	4
6	Windows Server IIS7	3
7	Ansible/Chef	2

Continued on next page

Table G.5 – continued from previous page

#	Tool	Usage
8	Helm	2
9	ServiceNow	1
10	Azure App Configuration	1
11	AWS Cloudformation Parameters/Jenkins Build Parameters	1
12	Ansible/SaltStack	1
13	Ansible/CFengine/Puppet/SaltStack/Windows Server IIS7	1
14	Ansible/Azure DevOps/Chef/Windows Server IIS7	1
15	Ansible/Chef/Puppet	1
16	HashiCorp Consul/HashiCorp Vault	1
17	Chef/SaltStack	1
18	Ansible/GitLab CI	1
19	Ansible/Puppet/Terraform	1
20	OutSystems	1
21	CoolConfigurator	1
22	Nomad	1
23	KeyVault/Octopus Deploy	1
24	Ansible/Terraform	1
25	Puppet/Windows Server IIS7	1
26	Jenkins	1
27	Ansible/Windows Server IIS7	1

Table G.6: *Testing tools used by participants.*

#	Tool	Usage
1	Selenium	16
2	Junit	5
3	Cucumber/Jmeter/Junit	4
4	Cucumber	4
5	Jmeter	4
6	Junit/Selenium	3
7	Cucumber/Junit	3
8	Cucumber/Junit/Selenium	2
9	Jmeter/Selenium	2
10	Cucumber/Selenium	2
11	Cucumber/Jmeter/Junit/Selenium	2
12	MOO/Microsoft Test Framework	1
13	Cucumber/Cypress/SpecFlow	1
14	Sonarqube	1
15	Junit/Selenium/Sigos	1
16	Junit/TestCafe	1
17	Jest	1
18	Azure DevOps	1
19	Dynatrace/Jmeter/Junit/TestNG	1
20	Mocha/PHPUnit	1
21	OpenTest/Selenium	1

Continued on next page

Table G.6 – continued from previous page

#	Tool	Usage
22	Dynatrace/Selenium	1
23	Selenium/TestNG	1
24	Cucumber/Jmeter/Selenium	1
25	Cypress/Selenium/TestNG	1
26	Cucumber/Jasmine/Jmeter/Junit/Postman Scripts/Selenium	1
27	Rspec/Selenium	1
28	Jmeter/Selenium/Splunk APM	1
29	Broadcom	1
30	HCL Test/Jmeter/Junit/NeoLoad/Selenium	1
31	Selenium/Tosca	1
32	PHPUnit	1
33	Cypress/Flood/Jmeter/Selenium	1
34	Behat/Cypress/Jest/Junit/PHPUnit/Psalm	1
35	Dynatrace	1
36	JOSE	1
37	Cucumber/Jmeter	1
38	Codecov/Cypress/Jest/Pytest/Selenium	1
39	Jmeter/K6/Selenium	1
40	Junit/TestNG	1
41	JOSE/Jmeter/Junit/Selenium/Tosca/UFT	1
42	Cucumber/Cypress/Junit	1
43	TOSCA	1
44	Citrus/Jmeter/Junit	1
45	Junit/Selenium/TestNG	1
46	Gatling/Jmeter/Junit/Wiremock	1
47	Jmeter/Junit	1
48	Cucumber/Dynatrace/Jmeter	1
49	Jmeter/Katalon	1
50	Cucumber/Rspec	1
51	Clayton/Junit/Provar	1
52	Cucumber/Jmeter/Karate/Selenium	1
53	Selenium/Xunit	1
54	Jmeter/K6	1
55	Jest/Junit/WebdriveIO	1
56	Cypress/Mabl/Selenium	1
57	Hypothesis/Jest/Pytest	1
58	Cucumber/Jmeter/Junit/Selenium/TestNG	1
59	Cypress/Jest	1

Table G.7: *Code analysis tools used by participants.*

#	Tool	Usage
1	SonarQube	38
2	SonarLint/SonarQube	6
3	Snyk	3

Continued on next page

Table G.7 – continued from previous page

#	Tool	Usage
4	JS hint	3
5	Jacoco/SonarQube	3
6	CheckStyle	2
7	SonarQube/VeraCode	1
8	ES lint/Visual Studio Code	1
9	CheckStyle/Clover/FindBugs/JS hint/Jacoco/SonarLint/SonarQube	1
10	JS hint/PHPCS/SonarLint/SonarQube	1
11	Cfn-lint/Cfn-nag/SonarQube	1
12	CheckStyle/FindBugs	1
13	Checkmarx/Stylecop	1
14	Gerrit/SonarQube	1
15	CheckStyle/Clover/Fortify/Gerrit/JS hint/SonarLint	1
16	Clover/Jacoco/SonarQube	1
17	Blue optima	1
18	JS hint/Jacoco/SonarLint/SonarQube	1
19	Rubocop	1
20	JS hint/SonarQube	1
21	Blue optima/SonarLint	1
22	Fortify/SonarLint/SonarQube	1
23	ReSharper	1
24	JS hint/Scrutinizer	1
25	OutSystems	1
26	Codesniffer/MessDetector	1
27	PHPCS/Psalm/SonarQube	1
28	Checkmarx/SonarQube	1
29	Boncode	1
30	Fortify	1
31	OS	1
32	CheckStyle/FindBugs/Jacoco/SonarLint/SonarQube	1
33	FindBugs/Jacoco/SonarQube	1
34	Checkmarx/Jacoco	1
35	Clover/FindBugs/Jacoco/SonarLint/SonarQube	1
36	CodeClimate	1
37	Fortify/Jacoco/SonarLint/SonarQube	1
38	CheckStyle/Fortify/SonarLint/SonarQube	1
39	Clover/SonarLint	1
40	Clayton	1
41	FindBugs	1
42	ES lint/Jacoco/SonarLint/SonarQube	1
43	Gosec/SonarQube	1
44	GHC/Mypy/Pylint	1
45	CheckStyle/FindBugs/JS hint/Jacoco/SonarLint/SonarQube	1
46	ES lint/Fortify/Jacoco/SonarQube	1

Table G.8: *Monitoring tools used by participants.*

#	Tool	Usage
1	Splunk	13
2	Datadog	6
3	New Relic	5
4	Nagios	4
5	Grafana/Prometheus	4
6	Prometheus	4
7	Graphite/Splunk	3
8	Azure Application Insights	3
9	Azure Webtest	2
10	Graphite	2
11	MONIT	2
12	Kibana	2
13	Humio/Nagios	1
14	Cloudwatch/Uptimemonitor	1
15	Azure Monitor	1
16	Icinga/Nagios	1
17	Elastic Stack/Graphite/Nagios	1
18	Cloudwatch/Alerta/Alertmanager/Elastic Stack/Grafana/Prometheus	1
19	Zabbix	1
20	Graphite/Nagios	1
21	Elastic Cloud/Kiali/Prometheus	1
22	AppPerfect/Ganglia/MONIT/Sensu/Splunk/Zabbix	1
23	Elastic Stack	1
24	Nagios/Splunk	1
25	New Relic/Prometheus	1
26	New Relic/Splunk	1
27	Elastic Stack/Splunk	1
28	Graphite/Nagios/Splunk/Zabbix	1
29	Site24x7	1
30	Elastic Stack/K8	1
31	Sensu	1
32	Appsignal/Datadog/Kibana	1
33	Broadcom	1
34	AppDynamics/Azure Application Insights/Azure Log Analytics/Splunk	1
35	Nagios/New Relic/Zabbix	1
36	Elastic Cloud/Graphite	1
37	Prometheus/Zabbix	1
38	Elastic Stack/Grafana/Graylog/New Relic/Zabbix	1
39	Grafana	1
40	AppPerfect	1
41	Prometheus/Splunk	1
42	OutSystems	1
43	logz.io	1
44	Elastic Stack/Kibana	1
45	Elastic Stack/OMI	1
46	Njams	1

Continued on next page

Table G.8 – continued from previous page

#	Tool	Usage
47	Elastic Stack/Grafana/Graphite/Prometheus	1
48	Azure Application Insights/Azure Monitor	1
49	Graphite/New Relic/Splunk/Zabbix	1
50	Elastic Cloud/Sensu	1
51	PRTG	1
52	Elastic Stack/Graphana/Prometheus	1
53	Datadog/Nobl9/Prometheus	1
54	Grafana/Loki/Prometheus	1
55	Cloudwatch/Nagios/Splunk	1
56	Datadog/New Relic	1

Table G.9: *Logging tools used by participants.*

#	Tool	Usage
1	Kibana	13
2	Kibana/Logstash	12
3	Logstash	11
4	Elastic Stack/Logstash	3
5	Graylog	3
6	Azure Application Insights	3
7	Graylog/Kibana/Logstash	2
8	Cloudwatch	2
9	Datadog	2
10	Humio/Logstash	1
11	Log4Net	1
12	Azure Application Insights/Kibana	1
13	Datadog/Rapid7	1
14	Beats/Kibana/Logstash/Syslog	1
15	Elastic Search/Fluentbit/Kibana/Logstash	1
16	Papertrail	1
17	Seq	1
18	Kibana/Logstash/Prometheus	1
19	Kibana/Log.io/Logstash/Sumo Logic	1
20	Kibana/Papertrail	1
21	Amazon EC2	1
22	Kibana/Logstash/SLF4j	1
23	Splunk	1
24	Azure Log Analytics/Splunk	1
25	Log4Net/SeriLog	1
26	Azure Monitor/Loki	1
27	Datadog/Logstash	1
28	Kibana/OutSystems	1
29	Elastic Search/Graylog/Kibana/Logstash	1
30	Graylog/Kibana/Logstash/New Relic/RSYSLOG	1
31	Log.io/Loki	1

Continued on next page

Table G.9 – continued from previous page

#	Tool	Usage
32	OutSystems	1
33	Log.io	1
34	Njams	1
35	Elastic Stack	1
36	Azure Application Insights/Python script	1
37	Kibana/Log.io/Loggly/Logstash	1
38	Loggly	1
39	Sumo Logic	1
40	Cloudwatch/Kibana/Papertrail	1
41	Datadog/LogDNA/Splunk	1
42	Loki/Promtail/Reluctantly/Stackdriver	1