Lwakatare, Lucy Ellen; Kilamo, Terhi; Karvonen, Teemu; Sauvola, Tanja; Heikkilä, Ville; Itkonen, Juha; Kuvaja, Pasi; Mikkonen, Tommi; Oivo, Markku; Lassenius, Casper

DevOps in practice

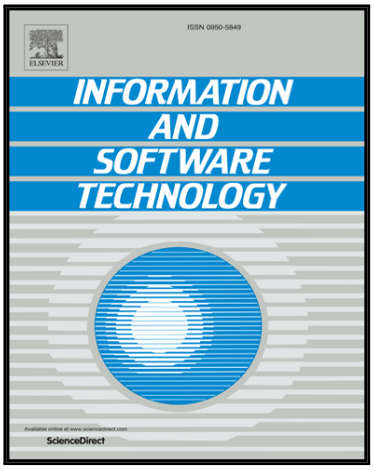
Published in:
Information and Software Technology

DOI:
10.1016/j.infsof.2019.06.010

Published: 01/10/2019

Document Version
Peer reviewed version

Accepted Manuscript



DevOps in Practice: A Multiple Case study of Five Companies

Lucy Ellen Lwakatare, Terhi Kilamo, Teemu Karvonen,
Tanja Sauvola, Ville Heikkilä, Juha Itkonen, Pasi Kuvaja,
Tommi Mikkonen, Markku Oivo, Casper Lassenius

PII: S0950-5849(17)30279-3
DOI: https://doi.org/10.1016/j.infsof.2019.06.010
Reference: INFSOF 6157

To appear in: *Information and Software Technology*

Received date: 31 March 2017
Revised date: 5 April 2019
Accepted date: 23 June 2019

Please cite this article as: Lucy Ellen Lwakatare, Terhi Kilamo, Teemu Karvonen, Tanja Sauvola, Ville Heikkilä, Juha Itkonen, Pasi Kuvaja, Tommi Mikkonen, Markku Oivo, Casper Lassenius, DevOps in Practice: A Multiple Case study of Five Companies, *Information and Software Technology* (2019), doi: https://doi.org/10.1016/j.infsof.2019.06.010

DevOps in Practice: A Multiple Case study of Five Companies

Lucy Ellen Lwakatare[a,*], Terhi Kilamo[b], Teemu Karvonen[a], Tanja Sauvola[a], Ville Heikkilä[c], Juha Itkonen[c], Pasi Kuvaja[a], Tommi Mikkonen[b], Markku Oivo[a], Casper Lassenius[c]

[a]*M3S, Faculty of Information Technology and Electrical Engineering, University of Oulu, Oulu, Finland*
[b]*Department of Pervasive Computing, Tampere University of Technology, Tampere, Finland*
[c]*Department of Computer Science, Aalto University, Helsinki, Finland*

Abstract

Context: DevOps is considered important in the ability to frequently and reliably update a system in operational state. DevOps presumes cross-functional collaboration and automation between software development and operations. DevOps adoption and implementation in companies is non-trivial due to required changes in technical, organisational and cultural aspects.
Objectives: This exploratory study presents detailed descriptions of how DevOps is implemented in practice. The context of our empirical investigation is web application and service development in small and medium sized companies.
Method: A multiple-case study was conducted in five different development contexts with successful DevOps implementations since its benefits, such as quick releases and minimum deployment errors, were achieved. Data was mainly collected through interviews with 26 practitioners and observations made at the companies. Data was analysed by first coding each case individually using a set of predefined themes and thereafter perform a cross-case synthesis.
Results: Our analysis yielded some of the following results: (i) software development team attaining ownership and responsibility to deploy software changes in production is crucial in DevOps. (ii) toolchain usage and support in deployment pipeline activities accelerates the delivery of software changes, bug fixes and handling of production incidents. (ii) the delivery speed to production is affected by context factors, such as manual approvals by the product owner (iii) steep learning curve for new skills is experienced by both software developers and operations staff, who also have to cope with working under pressure.
*Conclusion:*Our findings contributes to the overall understanding of DevOps concept, practices and its perceived impacts, particularly in small and medium sized companies. We discuss two practical implications of the results.

Keywords: DevOps, Continuous Deployment, Agile, Operations, Development

1. Introduction

DevOps, a portmanteau of *development* and *operations*, is an approach where software developers and operations work in close collaboration [1]. The goal is to improve communication and integration of development and operations in order to fully gain benefits of modern software development approaches [2] that employ rapid releases of new software features to, and subsequently learn from, end users [3].

DevOps concept, emerging in the context of rapid releases, was introduced in 2009 [4], however most of the problems it addresses were already identified in previous literature [5],[6],[7]. These prior works identified the problem of poor collaboration and lack of early involvement of operations team in software development process, which negatively impact software release time and quality in production environment [6], [7]. Much of the recent attention on DevOps comes after mainstream adoption of agile methods because most of them do not cover system in use (operations) life-cycle phase [8],[9]. As such, different concepts, such as *agile infrastructure* [10] and *boundary spanning* [11] were used to address the gap in agile literature but not systematically. For instance, agile literature on boundary spanning does not explicitly focus on operations unit rather on all units that interact with, and are external to, development unit [12], [11]. Early solutions to the problem have focused on establishing well-defined and formal procedures of involving operations in the development process [5],[11]. Through this, DevOps aims at improved delivery speed of changes to production and automation in software process [13].

Some DevOps practices can be found in prior literature [7]. Hamilton [7] presents a list of best practices that focus on automation when designing and deploying operations-friendly large-scale internet services to facilitate quick service delivery with minimum production incidents and administrative costs. To identify applicable DevOps practices, and gain a broader understanding of the nature and characteristics of its activities requires empirical investigation of how DevOps is enacted. Prior empirical research shows that DevOps is a multifaceted concept and its practices highly depend on how an organisation has interpreted the DevOps concept [14]. Studies that report successful adoption of DevOps, wherein the benefits of DevOps are realised, have shown that its implementation involves changes in organisational structures and roles [15], system architecture [16], processes and tools amongst others [17]. Details of the

*Corresponding author
Email address: lucy.lwakatare@oulu.fi (Lucy Ellen Lwakatare)

context that go beyond system domain [18] are necessary to inform and help determined the conditions in which the practices are best situated.

Deployment pipeline of software changes is where most of the cross-cutting concerns between software development and operations intersect [13]. Kerzazi and Adams [19] found that the DevOps engineer role includes infrastructure activities, such as deployment pipeline optimisation, among others. Deployment pipeline incorporates DevOps practices of automated deployment procedure and infrastructure-as-code [13],[20]. Implementing and integrating effective deployment pipeline to an existing software release process is not only challenging but requires diverse skill-sets, including that of operations staff [21],[22],[23]. Three anti-patterns of software release process are described by Humble and Farley [2] to include: deploying software manually, deploying to non-production-like environments during development, and configuring environments manually.

Besides well-known surveys [24], experience reports [22] and books [2], [25] from industrial leaders and practitioners, there are relatively few empirical studies on DevOps implementation in academia. This claim is supported by systematic literature reviews [20], [1] and a recent empirical study [26] that suggested there is very little to determine whether DevOps is indeed beneficial to Software Engineering. This raises the question of how DevOps is implemented in software companies besides the innovation leaders, particularly small and medium companies that have limited resources [27] and organisational boundaries [28].

The objective of this study is to present detailed descriptions of DevOps implementation in real industrial settings. Specifically, the paper explores the activities of the deployment pipeline of five cases from small and medium sized companies that are developing web-based applications and services. Three cases were developing software for an internal customer (other product teams) and the other two cases for an external customer (a public agency and an independent research organisation). The five companies were selected from a pool of organisations taking part in a larger Finnish national research project (detailed in Section 3).

The main contributions of this paper are threefold. First, it provides insights into the concept of DevOps and gives an enhanced definition of DevOps that is based on existing literature and interview data. Second, the paper provides detailed descriptions, including rationales, of DevOps implementation. Cross-case analysis of the implementation details are used to identify similar DevOps practices. Third, the paper provides detailed descriptions of perceived benefits and challenges of DevOps.

The rest of this paper is organised as follows: Section 2, presents literature review on DevOps. In Section 3, the research approach of the study is discussed. In Section 4, presents an overview of the cases, including the software development context and techniques. In Section 5, findings of the cross-case analysis are presented. Section 6 discusses the findings in light of prior literature, in addition to validity and limitations of the study. Section 7 concludes the paper with suggestions for future research.

2. Background and Related Work

Modern software development process is increasingly being characterised by frequent and rapid releases of software changes that enable fast feedback from end-users [3],[29]. The need to attain the capability to deliver software frequently, fast and in an automated manner as soon as changes are checked-in into the mainline is a major motivation for organisations to adopt DevOps [4] and continuous practices [3]. This section presents the DevOps concept and practices, as described in the literature. Empirical studies describing DevOps practices, benefits and challenges are presented to position this paper with the related work. As DevOps is intertwined with continuous practices, we only considered empirical studies that explicitly mentioned DevOps.

2.1. The DevOps concept

DevOps concept, pioneered by practitioners like Patrick Debois [10], aims at tackling inefficiencies in software development, release and operations processes that are caused by organisational split between the processes [9]. In such a context, the development and the operations units often experience conflicting goals of 'agility vs. stability' and face several challenges, including poor information flow and unsatisfactory test environments [6].

Multivocal 'grey literature' studies [30] and [31] found that terms such as 'movement', 'practices', 'culture', 'tool' and 'philosophy' are commonly used to refer to the DevOps concept. To minimize ambiguity, Penners and Dyck [32] proposed a scientific definition of DevOps as:*"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."*

The proposed definition by Penners and Dyck [32] has a consensus with that of [20] with respect to collaboration and communication between development and operations. Some scholars have argued against restricting DevOps to communication because it is vague considering that developers can use various sources of information to determine the requirements for software deployment and operations [33]. Further, it is argued that there is no established causal link between the different forms of collaboration and software release time or quality in production [13]. In similar line, according to [34], culture, which includes mind-set, should not be a defining aspect of DevOps; rather, the focus should be put on the engineering practices when defining the concept. The definition of DevOps proposed by [13] incorporates the later as it emphasises on speeding up the delivery of quality software that is achieved by employing a set of engineering practices from the time a software developer commits code to mainline up until when the code is deployed in production. Clearly as noted by [32] there is a need to make additional inquiries from practitioners¢¢ understanding of DevOps if we are improve the definition but research needs to go beyond that to investigate its actual behaviour (methods, practices and tools) and consequences [35]. This research makes an explicit inquiry of these aspects.

2.2. DevOps practices

DevOps concept is associated with both technical and non-technical practices. de França et. al. [31] divide DevOps practices into three categories. The categories describe common practices done jointly between the development and operations teams and practices distinct to development and to operations. The common practices category is further split into collaborative practices involving human interactions and procedural practices that are mostly automated. We elaborate more on the common practices in the procedural category herein rather than on those distinct to development and operations.

DevOps practices done jointly between the development and operations teams are incorporated into the deployment pipeline, as the focus is not limited to the design and implementation of system features, but it also includes the consideration of environments and tools that are used to support the development, deployment and operations of software features [13]. The deployment pipeline, according to Humble and Farley [2], is an automated manifestation of the entire software process constituting to all stages of getting software changes from version control until they are visible to end-users.

DevOps practices in the deployment pipeline involve automation of the deployment process, including automatic provisioning of the environments aimed at eliminating (or minimising) manual system hand-overs from development team to operations team. It may also be done to abstract the inner workings of the deployment procedures to developers, thus facilitating a low learning curve [23], [36]. In cloud-based systems, where the concept DevOps is often associated [37], the automated deployment mechanism is incorporated into the continuous integration (CI) server with pre-defined triggers to facilitate the automatic deployment of changes to virtual machines (VM) in production or other environments in the cloud [13], [25]. When software changes are available for release, a new VM image is created with the new system version in a process of baking an image. As part of the deployment process, configuration management tools are used to automatically provision, configure and upgrade existing VM with the newly created image while applying a selected deployment strategy [13], [38], [37]. Blue-green deployment and rolling upgrade are two commonly used deployment strategies in cloud-based systems [13]. The automated deployment mechanism can use different deployment strategies, but when created, it ensures the reliability and repeatability of system deployment [13], [38]. Several evaluative and experimental DevOps studies have focused on investigating reliability issues found by the tools when applying the different deployment approaches [39],[38]. Other studies have focused on finding better ways to integrate heterogeneous tools, their associated artifacts with the different deployment strategies [40].

In addition to deployment process automation, there are other practices that cross-cut the deployment pipeline and are of concern to both developers and operations personnel. According to Bass et al. [13], these include monitoring and security. Monitoring involves not only gaining insights from the various levels of the system stack [41], but also from the tools used in the deployment pipeline, as they have failure potential [13] and performance improvement opportunities [42]. Monitoring the execution of deployment scripts enable the detection of errors during deployments in addition to performing system health checks to detect issues and alert both developers and operations personnel [13]. Moreover, the deployment pipeline itself must be secured from malicious attackers and scenarios [13], [43]. This is in addition to ensuring security conformance during security audits [13], [44]. According to Schlossnagle [41], characteristics of successful monitoring includes well-articulated business goals; use of high capable tools and skillset to handle and interpret voluminous data; and data retention to foster culture of learning and understanding how failures transpire.

2.3. Previous empirical studies

This subsection presents DevOps practices, benefits and challenges from empirical studies published in academic forums. A summary of these is presented in contrast with our study findings in Tables 4, 5 and 6.

DevOps practices. DevOps practices include an automated system deployment mechanism to eliminate waiting times for environment provisioning [22], bridge the gap between development and operations [45] [46] and make system configurations reproducible [47] [48]. Tools such as Ansible, Chef and Puppet are reported to be used to facilitate automated provisioning of infrastructure [14][49], and makes transparent to developers the evolution of infrastructure and configurations [37]. Similarly, continuous monitoring through log aggregation and monitoring tools has become an integral part of the deployment pipeline, and are made accessible to developers [23] [16] [37] [44] [28].

DevOps benefits. Reported benefits of adopting DevOps include a reduction in the average release cycle time [22] [14] [50] (e.g., from two weeks to one day [45]), improved quality [14] [50] and eased tension and collaboration between developers and operations engineers [22] [16] [37] [14] [15], which also improved the morale of developers [45].

DevOps challenges. Reported challenges include difficulties in implementing an automated deployment process when software changes involve changes to database [45]. Another challenge identified was that DevOps adoption necessitates that both software developers and system administrators learn new technologies, tools and methods, in addition to performing their ongoing activities [34] [17] [28] [15]. Furthermore, when DevOps adoption is championed from the bottom up, senior managers are to be convinced of the benefits DevOps [17]. Significant efforts are required to automate the deployment process fully because it involves making changes to the ways in which an organisation manages infrastructure [22].

3. Research Methodology

This section describes the research methodology and steps taken in conducting this study. The study design, data collection and data analysis are further elaborated in the following subsections.

3.1. Study design and case selection

The study applies a multiple-case study approach conducted between January 2016 and January 2017. The multiple-case study approach was selected because it gave the possibility to investigate DevOps at multiple levels of analysis and in a real context [51], [52]. Furthermore, the case study approach fit to the purpose of our study, which is to identify what is happening and gain insights into DevOps adoption and implementation. This is in contrast to the objective of generating theory and cause-effect relationship, which would require considering other appropriate research methods such as Grounded Theory [53] and Experimentation [54]. Multiple case means five development teams experienced with DevOps approach were studied. The research questions used to guide this study were:

- RQ1. What is DevOps according to practitioners?
- RQ2. What key practices are employed in DevOps?
- RQ3. What are the perceived benefits and challenges of DevOps?

Case selection. A previous study identified that DevOps is challenging to adopt [55]. Based on this, we wanted to select companies that had successfully adopted DevOps from a pool of companies participating in the DIMECC Need for Speed (N4S) programme [1]. Five company representatives nominated themselves and allowed researchers to conduct the study in their respective companies. Their motivation to participate in the study was to share best practices amongst themselves. The selection of a specific team within the company was left to the company representatives. Background information of the cases and, later on, interviews with five practitioners from each case indicated saturation, i.e., no new viewpoints were emerging by adding new interviewees [52]. The unit of analysis was a software development team developing one or more software products or services for an internal or external customer of the company. The selected five cases are identified with the letters A-E, owing to the confidentiality that was agreed upon between the researchers and practitioners.

Design of the interview guide. Workshops were conducted by researchers to propose research questions, specify data collection process, identify roles suitable for interviews and develop an interview guide. The interview guide was developed in several iterations and reviewed by all researchers prior to data collection. The review of the interview guide provided a venue for researchers to request clarity or rationale for each question. The interview guide consisted of open-ended questions that were divided into six themes: 1) the background of the case and interviewee, 2) development practices, 3) build and integration practices, 4) monitoring and infrastructure management practices, 5) perceived impacts and 6) development culture. The full list of interview questions can be found in Appendix A. The outputs of the researchers' workshops were shared with the company representatives in a teleconference meeting at which a pilot case for data collection was selected and background information of the cases to be studied was requested.

[1] The Need for Speed (N4S) programme, executed by forefront Finnish software companies and universities, is a nationally funded programme aiming to create the capability for speedy value delivery based on deep customer insight (http://www.n4s.fi/en/)

3.2. Data collection

Data were collected mainly through interviews with 26 practitioners and observations at companies between March and May 2016. Table 1 summarises the roles of interviewees and observations made from the cases. The data collection was first executed in a pilot case (Case D), and three researchers conducted the interviews with practitioners. Data from the rest of the cases were collected by at least two researchers, where one of the researchers had participated in the pilot case data collection. The latter was done to ensure the lessons learned from the pilot case were considered during data collection in the other cases. The experiences and lessons learned from the pilot case were also shared in a workshop participated by all researchers prior to data collection for improvements in the interview guide.

Table 1: Interviewees and observations in cases

Case	Interviewees	Observations
A	Developers (3), project manager, site manager	Stand-up meeting, team working environment
B	Developers (3), product owners (2), director of culture and competences	Team working space, development technical environments
C	Developers (2), product owner, head of development, IT department-Ops team lead	DevOps meeting, team working environment, development technical environments
D	Developers (2), product owner, product owner of operations team, system administrator	Team stand-up meeting, company's business owners stand-up meeting, team working space
E	Developers (4), team lead	Team working space, ongoing deployment process.

At least five individual interviews were conducted in each case by researchers using the interview guide. The interviews were semi-structured with open-ended questions to allow researchers to make further inquiries based on interviewees' responses. Most interviews were done face-to-face on the company's premises, except for one interview, which was done via Skype. During the interviews, one researcher mainly asked the questions while the others took notes. All interviews were voice recorded and transcribed by professional transcription services. Each interview took, on average, 1 – 2 hours, except the interviews in Case E, which were considerably shorter, taking 30

Table 2: Description of the themes used to code the interview transcripts

Theme / subtheme	Description
Context / organisation	Data about organisation structure, roles and responsibilities, internal stakeholders, teams
Context / offering	Data about product or service, customer, external stakeholders, business
Feature flow / upstream	Practices employed before system implementation/coding begins e.g., road-mapping, requirements engineering, concepting
Feature flow / implementation	Practices employed by the development and operations team before and during deployment of software system to production
Feature flow / production	Practices done after deployment of software system to production
DevOps interactions	Explicitly mentioned interactions between development and operations
DevOps understanding	Explicitly given description of DevOps concept
Impacts	Perceived, real, expected effects of DevOps
Challenges	Challenges in the current and DevOps ways of working
Quality	Practices related to ensuring quality and non-functional requirements (performance, reliability, scalability, security)
Tools	Explicitly mentioned tools and their purpose of use before, during and after system deployment to production
Rationale and motivation of DevOps	Rationale given for adopting or using DevOps, including when DevOps was started, by whom and challenges that teams are solving through DevOps

– 60 minutes. Observations made on company premises also served as another important data source for triangulation. Observations were made, and notes taken, of events that took place on the days of the interviews.

3.3. Data analysis

The data were analysed using a thematic coding technique [56]. Data were first coded case by case and later on performing a cross-case synthesis, often used in multiple case studies [51] [52]. The analysis steps were done in multiple iterations in the NVivo [2] server project, accessible by all researchers from three different universities.

The first step of analysis involved the use of a thematic coding technique [56] to code the transcripts using a set of predefined themes. The predefined themes were developed by six researchers through a series of rigorous process steps that began with familiarisation with the data collected and pilot coding. The initial set of predefined themes was developed after discussion amongst researchers based on the interview guide and data collected. This was followed by two rounds of pilot coding to revise the predefined themes. The first round of pilot coding was done by six researchers who separately coded one similar transcript to determine a coding agreement level amongst them. NVivo can calculate a Kappa value of coding agreement between two sets of codes. As a rule of thumb, a Kappa value of 0.4-0.75 describes a fair to good agreement, whereas a value over 0.75 describes excellent agreement. After the first round of pilot coding, poor coding agreement values (Kappa less than 0.4) were observed in most themes (8 out of 11). Subsequently, the researchers had a one-day workshop to compare how different researchers had coded the individual themes and discussed their rationale. By the end of the workshop, the researchers had created a coding guideline document containing a list of basic principles and a codebook listing the predefined themes and their descriptions, showin in Table 2. To ensure the coding practices had improved, researchers selected another transcript that was coded in its entirety by five individual researchers. After the second round of pilot coding, the researchers were satisfied with the level of coding agreement. The task of comprehensive interview coding was distributed amongst the six researchers. Each researcher coded at least four interview transcripts. Weekly teleconference meetings were held to check the status of the coding process and discuss any emerging issues.

Following the completion of thematic coding, case-wise summaries of each theme were written and subjected to a cross-case comparison using the cross-case synthesis method. Cross-case synthesis was done following recommendations provided by Yin [51]. Yin [51] recommends the "*creation of word tables that display individual cases according to some uniform framework* ". In this study, tables (one for context subthemes and another for feature flow subthemes) were created in Google Sheets to display case data using a framework developed from literatures by [27] [2] [13]. The framework described the activities that are executed as code changes move through the deployment pipeline and while in production and, in addition, the rationales given by interviewees for having (or not having) a particular practice. The parts coded with benefits were read and

[2] NVivo is a qualitative data analysis software.

benefits were added to a table case-by-case and further thematically grouped cross-case.

4. DevOps implementation and adoption in each case

This section presents information about teams, processes and environments used to develop and operate software applications and services in each case. Table 3 gives a summary of the development context and Figure 1 shows the deployment pipelines of each case. Three cases were end user-facing services, of which the longest running service (Case C) had been in production for over a decade. The other two had been initially deployed between 2015 and 2016 (Cases A, B). At the time of the interviews, four cases (Cases A, B, C, F) were using the Amazon Web Services (AWS) cloud platform and Case D was transitioning to AWS from an internal private VMware-based cloud.

4.1. Case A - Road maintenance reporting tool

The company behind Case A mainly provides digital business consultancy and solutions to its customers. Case A provides product-engineering services to a public sector customer in a fixed-price project that started in 2014. The tool is a web-based application that displays information about road maintenance activities gathered from various other systems and is meant to modernise an existing solution with more advanced features, such as real-time follow up and map services. Case A consists of six 'full stack' developers and one project manager co-located at their company's premises. The company has a matrix organisation structure wherein there exists project teams whose members also belonged to other focused teams, such as Java team and integration team. The company has a separate operations team that is managing company's third party provided cloud infrastructure and is located in another city and one member from the team interacts with Case A members.

Techniques and environments. Product requirements were specified at the beginning of the project and stored in a product backlog that is maintained by the project manager in Jira. Developers pick tasks from the backlog and implement new features by branching from master branch to a development server. New software changes are first tested by the developer and then reviewed by another while in development server. Once code-review is completed, accepted changes are merged to the master branch. New merges to the master branch trigger the CI server to automatically build and subsequently deploy to the test environment, where unit tests and end-to-end tests against test database are executed. If the tests pass, every morning at 6 o'clock, the application was automatically deployed from CI to a staging environment, that is accessible by the customer. The project manager and the customer perform additional manual exploratory tests in the staging environment. Case A has a separate repository for the Ansible projects, which are used for setting-up test, staging and production environments automatically. Access to management console and APIs of the infrastructure is managed by the operations team. Case A members interact with operations team through HipChat, particularly when setting-up new environments and accessing monitoring logs from CI server especially during build failures. For configuration changes that are commited to the repository, Jenkins automatically runs the playbooks and builds them afterwhich the artefacts are archived. The developed product is one JAR file that gets deployed to the server using a playbook, that copies the configuration template and the JAR file to the server, and then calls a system restart.

DevOps introduction and adoption. DevOps approach in Case A was largely introduced by developers who had the autonomy and flexibility to improve their ways of working. According to one developer, the need to improve development workflows was one motivation for adopting DevOps:

> 'First tasks I've been involved in when I joined were DevOps like working habits or techniques. I've been like a consultant in that role...one of the first things I did was to set-up a development environment that is very easy to use because it was like Wild West...there were no Git workflows and the development environment was installed painstakingly in every workstation...So we switched to Vagrant...Git-flow and set-up six private development servers for each person to develop features by branching from the main branch'.

Prior to this improvement, the team spent huge efforts in merging code and resolving merge conflits, which were causing broken builds often. Case A had acquired external cloud resources— not managed by operation team— for provisioning the shared development environments, and the team had full access to infrastructure, including management console. The latter was supported by the customer and operations team.

4.2. Case B - Occupational Health and Safety Services

The company of Case B specialises in providing digitalisation services to business customers. Case B provides product-engineering services — software development, technical expertise and infrastructure — to an independent occupational health and safety research, development and specialist organisation. Development work is carried out in a project that started in early 2015, with the goal of implementing the organisation's digitalisation strategy to provide more occupational health-related services online. Case B is developing three web services, including websites about safety at work, occupational health and a quality portal. Case B has five 'full stack' developers, one technical project manager and one user experience (UX) designer co-located at the customer's premises with two product owners employed by the customer organisation. At the time of the interview, the company had two business units: architecture office and construction office. The architecture office focuses on system requirement definition including high-level architecture specification, while system implementation was done by the construction office. Case B has no separate operations team, rather the required expertise is said to be embedded inside the team through developer self-learning or discussion with domain experts.

Table 3: Summary of case context description

Aspect	Case A	Case B	Case C	Case D	Case E
Company type and size	Digital business and service consultancy company, 600+ employees	Digitalisation service company, 300+ employees	Digital marketing and sales solution company, 300+ employees	Public service broadcasting company, 2500+ employees	Cyber security experts and solution company, 1000+ employees
System	web-based road maintenance reporting tool	websites for safety at work, occupational health and quality portal	web-based service channel for companies to order, use, monitor and manage their purchased services	REST API for media content	Security cloud services
Customer	National Transport Agency	Research organisation in the field of occupational health and safety	Product teams inside the company	Product teams inside the company	Product teams inside the company
End users	National Transport Agency, Regional transport and environment centres and road maintenance contractors	Occupational health providers, researchers, employers and all people interested in occupational health topics	Companies	Product team, external developers	Corporate customers and external developers
Team size	6 'full stack' developers, 1 project manager	5 'full stack' developers, 1 project manager, 1 UX designer	4 developers, 1 UX designer, 1 product owner	7 developers, 1 product owner	8 developers, 1 team lead
Release cycle time	Fixed schedule project of 2 years	2 weeks	Not defined	Not defined	1–2 weeks
Deployment frequency	Daily to staging	Several times to production during two-week sprint	Daily to production	Daily to production	Daily to Alpha and weekly to Beta environments
Build output	JAR file	Docker image	AWS machine image	Debian package	AWS machine image or Docker container
Tools	Deveo, Jenkins, Selenium, Ansible, GrayLog	GitHub, Jenkins, SonarCube, Ansible, Docker, Amazon CloudFormation, Amazon monitoring services	Bitbucket, Jenkins, Chef, New Relic	Git, Jenkins, Puppet, New Relic	Bitbucket, Jenkins, Docker, Amazon CloudFormation; Kinesis
Languages	Clojure	Java, JavaScript, Node.js, Python	Java, Scala and Node.js	Clojure, Scala	Python, Java and C

Techniques and environments. A list of user stories (tickets) is maintained by the product owners. During each two-week sprint, estimated tickets are kept in priority order in a sprint backlog on the team¢¢s Kanban board. On Kanban board there are also *'infrastructure or operations'* related tickets, in addition to the user story tickets that are kept as small as possible to implement in shortest time. Tasks are implemented on feature branches and then merged to the master branch in GitHub. The CI server tracks changes in the master branch to automatically build a Docker container image. Unit tests are executed as part of build process and SonarCube is used for checking code coverage. The image is then automatically installed from CI to development server. Once in development server, code changes are reviewed and application tested by another developer, if there is time. Product owner reviews implemented tickets and perform manual front-end tests in the development environment. With the permission of the product owner, accepted changes/tickets are deployed to an acceptance testing environ-

ment by push of a button. Because the team uses microservice architecture, components that relate to accepted tickets are first selected from the different build pipelines. When in acceptance testing environment, the product owner announces the availability of new features for testing by other stakeholders in the client organisation. Again, with permission from the product owner, the developers deploy to production, which maybe done several times during (or at most in) a two-week sprint cycle time. In addition to using Amazon CloudFormation templates, Case B uses Ansible scripts, which are stored in a version control where everyone has the possibility to modify them, to automating deployment and the provisioning of infrastructure.

DevOps introduction and adoption. Case B's customer project is considered a prime example of a successful implementation of DevOps practices, made possible by the customer's willingness to adopt an AWS infrastructure and diverse skillest embedded into the team. According to director of competence and culture at the company and developers in Case B, adoption of DevOps in customer projects varies and it has not been adopted in many customer projects of the company. Adoption of DevOps is considered to be difficult when customer has an external operational service provider as well as multiple suppliers due to problems such as development team's limited access to production. Specific to Case B, DevOps adoption is elaborated in the following statement given by a developer:

> 'We've made some small changes. But not in such a way that we decided that now we'll start using DevOps. [our company] has guys who have a background in things like that, in fact, some have been involved in this project as support. So when we face a situation that there is a problem with infra and we don't have experience on that we can contact these guys or they can come to the site and help if the customer allows it. And that's how it was before the launch. We had one traditional sys admin person. But they were a part of the team, showed us how these things should be done'

4.3. Case C - External Business Customer Access Point

The company of Case C offers digital marketing and sales solutions that provide timely business and contact information. Case C develops and maintains a web application serving as a self-service channel for companies' to order, use and monitor their purchased services. For the business customers, the application acts as a control panel to acquire and manage different services ordered from the company of Case C. Case C consists of four developers and one UX designer co-located together with a product owner. The company's development unit has altogether 22 software developers assigned into five different software development teams. The developers are hired from different consulting companies. The company has a separate operations team (IT organisation) consisting of four members, two of which are software developers. Their responsibility is to manage the infrastructure, including cost and user rights management, building automation and giving support to development teams.

Techniques and environments. The product owner maintains and prioritises development tasks in a product backlog. Tasks in the backlog are kept as small as possible, so they are easy to develop and push to production. Developers can freely pick tasks from the backlog and begin work in local branches. Code commit to master branch in Bitbucket trigger CI server to build application package. After the application package is built, it is first promoted and then deployed to subsequent environments by a manually clicking a button click in CI. The deploy to test and production buttons execute CI job, which deploys the application to test or production environments. The production deployment job starts a new instance, installs the required software and security updates, creates a AWS image and starts a new instances based on that image. Chef is used for infrastructure automation.

DevOps introduction and adoption. The introduction of DevOps into the company was motivated by the business need to reduce their services' time-to-market. As such, the transformation towards DevOps started in 2011, when the company decided to move from on-premise infrastructure to a cloud-based infrastructure offered by Amazon. The use of AWS and its underlying technologies minimised the need for a large separate operations team, as software deployment and infrastructure management, including configurations, could be done also by the development team once the automation in deployment pipeline was implemented.

4.4. Case D - REST APIs

The company of Case D is a media company, offering radio and television programmes across all telecommunication networks. The company is also offering broadcasting programmes and other content services online. Case D develops REST APIs that are plugged into the company's front-end web services. Almost 20 APIs are developed and maintained by Case D. The development team consists of seven developers, hired from consulting company, whose number may rise to about 15 developers at peak times. In addition to other development teams, the company has a separate operations team that is building and supporting company's VMware infrastructure. The operations team consists of five members led by product owner, who also serves as a software architect in the company.

Techniques and environments. The developers have one task at a time in progress. No work estimation or process tracking is used. The team¢¢s progress is visualised with a Kanban board managed with daily meetings. There are no specific release cycles. There is trust that small changes will not break anything and if they do, they will be fixed. Freeze periods are used to ensure functionality of the services during holidays and weekends. Every piece of code ready to be put into the master branch in the version control system is reviewed by a second developer to ensure code quality and to maintain coding guidelines. Commits to the master branch, kept deployable and working, are done daily. A CI server validates the code with unit and integration tests and stores a Debian package to an internal repository. The build status is monitored on a radiator

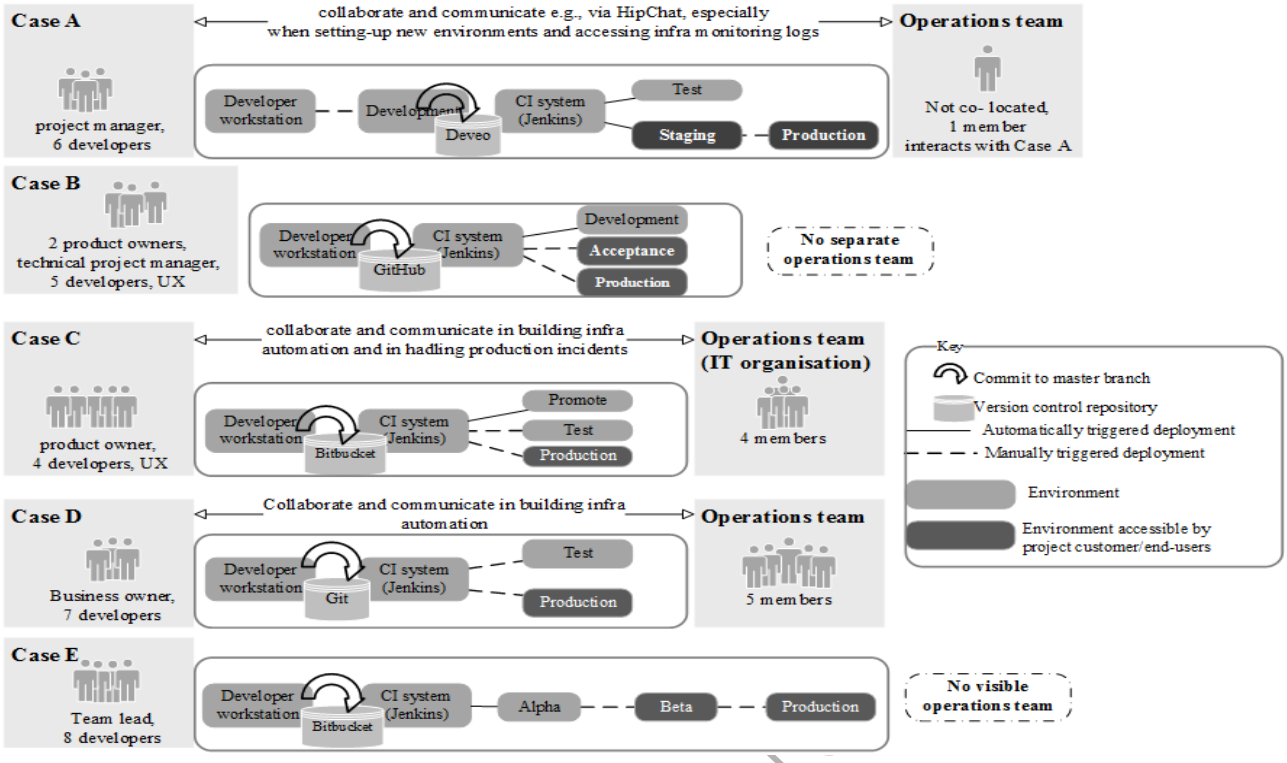


Figure 1: Deployment pipelines of studied cases

at the team's premises. From the repository, version-controlled deployment scripts built in-house are manually triggered in order for deployment to test or production environment. Puppet is used to provision the environments.

DevOps introduction and adoption. The introduction of DevOps in Case D was largely driven by enthusiastic practitioners within the company. Adoption of DevOps practices into the entire company has not yet been realised. However, DevOps is reported to be infiltrating across the different parts of the company mainly due to enthusiastic practitioners who are considered as change agents to company's ways of working. According to one of the enthusiastic practitioner, the first step taken in DevOps adoption was giving development teams access to the production environment after ensuring all necessary legal requirements were met. This was followed by the team gaining the ability to manage configurations and implement infrastructure codes of the APIs. Much of the DevOps transformation is further motivated by the company's decision to move to the AWS cloud in the near future.

4.5. Case E - Background Services for Products

The company of Case E is a European cyber security company providing antivirus, security software and VPN services. Case E cloud team develops cloud-based background services that are used by products developed by the company and its external partners. The services have different purposes, use different technologies and are developed with different programming languages. Typically, the individual services are stateless and not large, but may need to handle a large number of simultaneous requests. Case E consists of eight developers and a team leader. Most team members are highly experienced, have worked in the different parts of the company for several years, and posses a diverse set of technical skills. The team is considered at the cutting edge of development methods and technologies.

Techniques and environments. The requirements are recorded in a ticket-tracking tool. Development is done in feature branches that live no more than two weeks to avoid merge conflicts. After passing the local (unit) testing, a pull request is created to review and make possible changes. Once completed, the pull request is merged to the master branch and automatically built in the CI. Once built, the code is deployed on an alpha server and monitored. Client product team members do the alpha testing, typically using test automation. Some services have system tests that are executed in the AWS environment. If all performance and stability patterns look good, the code is pushed to a production server as a beta service for a limited user base and, if everything goes well, finally to general use. The cadence of beta releases depends on the maturity of the system. It may be two weeks for mature services or a few days for new services, but critical fixes may be completed and published in one hour or less.

DevOps introduction and adoption. The organisation has been gradually transitioning to DevOps and specific to Case E, it involved them taking the responsibility for the deployments and operations of those services. One motivation for introducing

DevOps according to team lead was breaking the aligning goals of development and operations:

> 'It originated roughly four years ago. We had operations team and systems development team. Those were working really close with each other, but also teams were having different goals and targets. Success and failures were measured in different ways. Operations was more about keeping systems up and not disturbing anything, and on the development side it was about speed of development and getting things into production. Those were competing goals and that created friction. So we actively started breaking those things'

5. Results

This section presents findings of cross-case analysis that identifies similarities in the descriptions of DevOps concept, practices and perceived impacts. Table 4 summarises common observed DevOps practices and other software development practices. Commonly perceived benefits and challenges are summarised in Table 5 and Table 6 respectively. From the perceived impacts, attention is given to the explanations since they are generally applicable and thus difficult to argue as a mere result of only applying DevOps.

5.1. A shared understanding of the DevOps concept (RQ1)

DevOps, according to the majority of the interviewees, means ownership and responsibility of software development team to design, implement, test, deploy and maintain in production the web applications and services. The main goal is to make software changes visible in production fast was to allow quick feedback of the changes. Operations personnel support development teams in activities related to automation of deployment pipelines, sharing knowledge of security, scalability and performance issues, and jointly resolving incidents. Interaction between software development and operations personnel was mostly informal and on as-needed basis (A, C, D).

> 'DevOps means the team has the complete control over the infrastructure, and the deployment and, every possible thing that can be automated is automated' (Developer, Case A)

> 'DevOps in my opinion is that, persons who are developing the applications are also the same ones to maintain them in production. If something goes wrong then the same persons are investigating the problems and fixing them' (IT department-Ops team lead, Case C)

> 'DevOps is when you have full responsibility of the software you are developing in production ' (PO of operations team, Case D)

> 'It means giving the team more responsibility and control...that you don't handover to operations your deployment. It's gaining confident in your deployment procedure and your ways of working' (Developer, Case E)

Culture and mind-set. For most of the cases, the culture was that of *trust, team empowerment and cooperation*. In Cases B, C and D, development culture was quite inclusive and open to learning. There was a recognition of value in having developers who were consultants from various companies, co-located with the customer and other members responsible for service development. This enabled frequent interactions, information flows and building of trust and team spirit, which were seen essential in DevOps. In all Cases, the developers had the freedom and management support to learn, choose, improve or influence suitable development practices in the project. At the company level, there was a strong culture of sharing knowledge in training groups dedicated to different themes, e.g., a DevOps training group where members meet and discuss topics, problems, etc. related to that theme —similar to community of practice in agile.

In all cases except Case A, the development mind-set was that of getting *features to production as fast as possible and fixing them later on*. In Case D, the head of the API team especially emphasised delivery speed over processes, bureaucracy and even quality. Delivering early and often, diving into production and fixing later on describe their DevOps mind-set. In Case C, the mind-set emphasised the ability to take risks while allowing failures without blame. This was visible in frequent production deployments, in addition to focusing on building a deployment pipeline that allowed reverting and corrective actions rather than the comprehensiveness of, e.g., test suites. This was also evident in management's emphasis on people making decisions, acting fast and correcting direction later rather than waiting and planning. In Case E, the mind-set was to get something useful to the customers fast. Thus, development focused on working on small features and getting them into production as soon as possible. Contrary to the other cases, the mind-set in Case A, also supported by management, was to deliver on time and with good quality instead the 'deploy and fix later' mind-set. The development team greatly valued quality and focused on ensuring high quality before presenting changes to customer.

5.2. DevOps practices situated in context (RQ2)

5.2.1. Automated deployment mechanism

Deployment scripts facilitated the automatic installation of software to a target environment by software development teams. Deployments scripts were mostly triggered from the CI system. Software developers in all cases had access and could modify the scripts, which were also version-controlled. The access and visibility of the deployment scripts by software developers was seen positive and ensured knowledge transfer to developers about the deployment process.

Deployment to a non-production environment was mostly automated after the completion of a build while to production it was mostly manually triggered as it involved getting confirmation from product owner or project manager (cases A, B). Production deployments could be executed by any developer, but often only after notifying other team members of such an intention (Cases C, D, E). The notification served the purpose

Table 4: Key practices employed by the studied cases

Key practices	Findings of the study	Supporting empirical DevOps literature
DevOps practices found in deployment pipeline		
Automated deployment	Deployments triggered from Jenkins (Case A, B, C, E) and using a company-tailored shell script (Case D)	[22] [37] [45], [14] [49] [46]
Infrastructure as code	Version-controlled provisioning scripts or templates for management and configuration of infrastructure using Ansible (Case A), Chef (Case C), Puppet (Case D) and Amazon CloudFormation (Case B and E)	[47] [48]
Continuous monitoring	Monitoring of system health checks by developers using tools such as New Relic, Kinesis and Amazon monitoring tools and communicated via radiators, email alerts, PagerDuty and chat applications	[23] [37] [44] [16] [28] [57] [45] [14]
Existing key development practices		
Trunk-based development	Frequent merges of branches to the mainline in all cases to minimise effort and conflicts resulting from code merges. In Case D, feature branches were released and thus maintained for some APIs	
Change-based code review	Code reviews done prior to merging code to the mainline in Cases A, D and E and after merging to the mainline in Case B. Main purpose was to ensure quality and an opportunity to share information about coding styles and standards. Factors in code review considered important included knowledge of the reviewer, size of code changes to be reviewed and review environment support	
Continuous Integration	CI server to execute unit tests and build system into deployable packages was implemented in all cases using Jenkins. Additional automated integration tests for some specific software components was done for Case E. Radiators are used to communicate the status of builds	
Agile, lean practices (and their adaptations)	No explicit agile method in cases, except for Case B, which was using Scrum. However, implicit adaptation to agile Scrum and lean Kanban. From Scrum practices, e.g., daily/weekly stand-up meetings, retrospectives and task estimation. From lean, task tracking and a Kanban board were used in all cases. Rationale for not using Scrum in its entirety was to be more efficient and less constrained to the procedures	

of avoiding duplicity. For example, in Case E, the team considered it important to avoid baking another image immediately after a previous one had just been created.

The cases employed different deployment strategies, such as blue-green deployment (Cases C and E), rolling upgrade (Case B) and canary deployment (Case E). Roll-back mechanism was said to be in place in Cases B, C, D and E. The deployment procedure was not explicitly documented except in Case B, where the procedure was documented in a Confluence wiki. In Case E, some software developers, particularly new members of the team, documented the deployment steps for themselves.

Automatic deployment to production environment from code commit for web services that were developed primarily for other product teams (Case D, E) did not extensively rely first on approvals from product owners as compared to the contracted web applications incorporating user interfaces (Case A, Case B). Canary deployment used in Case E ensured testing in production environment with the users.

> 'Features that change the output or bring some new data or expectations we usually inform the guys, who have ordered it, to try it out in testing because they might have different tools and things that they test '(Developer, Case E)

For the contracted web applications, PO or the customer needed to approve almost all changes often done at the end of the sprint and before deployment to production environment.

> 'There's some room for improvement on my part so that I could react to, and test, the features in the testing environments more quickly. Of course, neither of our product owners is a full-time product owner who could completely focus on the task. So if a few post-its remain and I don't have time to test them on the same day, one could think about some kind of a back-up arrangement'(PO, Case B)

Tooling and culture of team empowerment, trust and teams¢¢ shared responsibility for the ownership of the developed web applications and services observed in teams ensured that software changes to production are delivered quickly..

> 'It has been almost five years we have worked that developers are responsible also to production. If there is something wrong, somebody will participate in solving the problem and, developers have all the required permissions to access production and deploy new versions'(IT department-Ops team lead, Case C)

5.2.2. *Infrastructure-as-code*

Provisioning and configuring of environments repeatedly and reliably was done as a part of the deployment process using tools, such as Amazon CloudFormation (Case B,E), Chef (Case C), Puppet (Case D) and Ansible (Cases A and B). Docker was used in automated deployment procedure for some services in Case E and was trialed on some selected products in Cases C and D. Docker (Cases B) and Vagrant (Case A) were used to provide virtualisation in the local development environments.

The use of IaC on AWS cloud environment (Case B, C and E) and VMware cloud environment (Case D) provided an important feature, which is immutability of resources. This meant that environment resources, such as instances of production environment, could be newly provisioned and quickly set-up using machine image (Case B, C, E) or scripts (Case D). This minimised human error and ensured predictability and reliability of application deployments.

> 'Automation of server installations we have been doing enabled to have development environments the same. Most are made such that they are easy to replace and build. You can rely that you can build any server again if something horrible happens. It's also quite nice because, otherwise, I wouldn't be able to maintain these many servers' (Sys admin, Case D)

Networking services to the virtualised resources required some careful consideration. It was reported by a software developer of Case E that Docker, a technology for lightweight virtual machine, was used when working on network services, which simplified operations. In Case D, both software developers and operations staffs reported of poor automation of network services in infrastructure was affecting the deployment procedures.

> 'If we are working on network services, for these we often have Docker-based system'(Developer, Case E)

> 'The network has been a bit flaky at times. That's something we cannot control and have to rely on outside help to resolve'(Developer, Case D)

In Cases A, C and D, provisioning scripts were initially developed by operations personnel who then made them accessible to developers. In Cases A, C, D and E, the provisioning scripts were version-controlled and in Cases A, D and E, changes to configuration parameters were sometimes reviewed by another developer (Case E) or system administrator (Cases A, D). While provisioning scripts or templates once created are reused and that information for editing them is readily available from documentation, software developers saw it as a competence of operations staff which overtime they could acquire through learning and interactions with operations staffs.

5.2.3. *Monitoring as a continuous activity performed by the development team*

Developers were actively monitoring the web applications and services they developed, both in production and in CI environments. Monitoring was performed for various purposes, including providing visibility to the success or failure of the deployment and quality of web applications and services (Cases B, C, D, E). Development teams monitored the systems in production to be sure they were running, as well as to receive information on the usage behaviour of the services (Cases C, D, E). Monitoring tools, such as New Relic, Kinesis and Graylog were used to provide monitoring automation, and the tools were configured to alert the teams in case of incidents or anomalies. Monitored data were provided in the form of graphs with patterns and trends displayed on wall radiators found at the team¢¢s workspace (Cases D, E) or on dashboards accessed via the developers' workstations (Case C). Outside working hours, on weekends and public holidays, the teams in Cases C, D and E did not report to have a formal process for monitoring the services.

Alerts of incidents were sent to both operations staffs and developers using tools such as PagerDuty (Cases C, E, D; planned). Both developers and operations were responsible for reacting to alerts (Cases C, D, E). In Case C, one developer (on weekly rotation) was assigned to be on call and was responsible for reacting to production alerts. The role rotated amongst all developers and if the developer assigned to be on call was not available, the alerts were sent to all developers. Monitoring activity was observed to be mostly established for the web services developed internally (Case C, D, E) and least practised in contracted web applications (Case A, B).

> 'We have a department that has information screens about the hosted servers, and they react when the disc space is going to run out. If we had a standard setup that would be integrated to monitoring services, we could use some kind of Ansible role, to setup the servers so that they report to the monitoring services and the team here would see the monitoring data. But, the monitoring is completely, in the hands of the IT' (Developer, Case A)

5.3. *Perceived benefits and challenges of DevOps practices*

5.3.1. *Perceived benefits*

B1. Improved delivery speed of software changes. An improved speed in the delivery of software changes was the most commonly perceived benefit of DevOps. Cases B, C, D and E were able to implement and deliver features to production in shorter time periods with longest being two weeks. At best, the speed was reported to have improved from months to days. Improved speed made it possible to get fast feedback from production environment. The employed DevOps practices were seen as enablers and could facilitate continuous deployment of changes whenever necessarily e.g., during bugs fixes.

> 'We have automated processes. Whenever we implement something new, it doesn't take too much time for the customer to get and test it' (Developer, Case A)

> 'A few years ago we had new versions of product out in production four times a year and then once a month. When we started to use DevOps this was done on weekly and daily basis. Most important

Table 5: Perceived benefits in DevOps

Findings of perceived benefits from this study	Finding from literature
B1. Improved delivery speed of software changes Improved speed in the development and deployment of software changes to production environment	Improved release cycle time [22] [45] [14] [50] [46]
B2. Improved productivity in operations work. Decreased communication problems, bureaucracy, waiting overhead due to removal of manual deployment hand-offs and organisational boundaries; Lowered human error in deployment due to automation and making explicit knowledge of operation-related tasks to software development	Eased tension and increased collaboration between development and operations [22] [37] [16] [45] [14] [15]
B3. Improvements in quality. Increased confidence in deployments and reduction of deployment risk and stress; Improved code quality; Improved product value to customer resulting from production feedback about users and usage	Improved software quality [45] [14] [46]
B4.Improvements in organisational-wide culture and mind-set. Enrichment and wider dissemination of DevOps in the company through discussions and dedicated training groups 'communities of practice'	

thing is that we are getting things done, quickly' (IT department-Ops team lead, Case C)

'We deploy into production daily and try to keep the tasks small so that we can deliver those quickly and often' (Developer, Case D)

B2. Improved productivity of operations work. All cases had perceived the benefit of having removed manual steps and hand-offs in deployment process. Full ownership of the developed web application and service by software development teams was reported to remove barriers, bureaucracy and waiting overhead (Cases A, B, C, D). Developers feel less frustration due to the reduction in arbitrary policies, waiting, inefficient tasks and developing wrong features. Cases B, C and D reported automation as a way to create documentation and to remove tacit information in operations-related tasks, thus making knowledge explicit and transparent to developers. As a result, the teams relied less on external experts while also lowered the risks of human error in deployments. Three cases (Cases B, C, D) perceived a lessened maintenance workload for an operations team, as it was shared with a development team. This was mostly facilitated by tools in infrastructure automation and the use of the cloud-based virtualisation of the server infrastructure. Thus, the responsibilities of the operations teams shifted from server maintenance to developing the infrastructure tools and automation that supported the development teams.

'At the company, up until four years ago, we had this traditional approach that there was a team that made software and another team that handled the operations. So the biggest change was that we empowered development teams, to run their software in production and we kind of scraped that whole development and operations barrier' (PO of operations, Case D)

B3. Improvements in quality. All cases perceived DevOps to increase production quality and reduce deployment risks and stress. Three cases (Cases B, C, D) further mentioned that DevOps directly improved software quality due to high confidence in deployment to production, which in turn was facilitated by the high degree of automation and the capability to roll back the changes. Automation in testing and deployment contributed to the positive impact on the overall quality of the production code. Deployment in small increments increased production quality, as the quality and risks of small changes were easier to control than were those of large changes with long deployment intervals. The ability to make fast deployments in small increments, together with the fast feedback cycle, further shortened the lead-time of fixes for production problems. Feedback from production provided transparency about the product on the market, the customers and the users of the product to development teams. Faster feedback guided development more rapidly towards and allowed the product management and customers to follow the progress of in-production features in a realistic way (Case B, C, D, E).

'There is the feeling of security of what we push into production. We push into production in such small chunks that it is possible to reliably test that everything is okay. In the previous projects, I have always been nervous about that since so many changes were pushed into production at once' (Developer, Case B)

'Two biggest points are speed of development and quality of the software...The quality of the code and services is going up in a very significant way. That is usually shown in customer feedback and our incident amounts have been falling rapidly' (PO of operations teams, Case D)

B4. Improvements in organisational culture and mind-set. DevOps culture and mind-set, which were enriched with colocation, were observed in the wider dissemination of DevOps approach across the organisation. In Case A, a wider dissemination of knowledge related to DevOps was made possible through training groups 'communities of practice' across the company. Case B saw colocation of the development team with the customer to be an enrichment of the organisation and in support of the DevOps mind-set. Case D saw the benefit on a wider

organisational level, where other teams had started to learn and copy DevOps practices from the team, and things were getting done fast. In Case E, the mind-set was shared and supportive towards the DevOps approach, both from management and development team members.

5.3.2. Perceived challenges

C1. Insufficiencies in infrastructure automation. In Cases A and D, provisioning scripts were considered error-prone and, according to developers, they did not work in some environments. Moreover, in Case D, the automation of the network in was said to be difficult in addition to dealing with legacy system. In Case C, the data platform handling customer data was causing the team to display erroneous reports to end users, as it was not being updated correctly. The data platform was being managed by the operations team and, according to the head of operations, the problem was due to incorrect updates to service index of search engine data that is loaded every night. As this was impacting the quality of service, meetings between operations and development were frequently held, one of which was observed by the researchers of the study.

> 'Networks are pretty hard. Some of the databases are pretty hard too because the old relational databases haven't been designed to be clustered' (sys admin, Case D)

> 'Every single week, some of the reports are missing data. Data run has stopped during the night because system has run out of memory or something like that. Then you always have to manually do so much work and check if you have all the data there or not, and has been, incredibly frustrating time' (PO, Case C)

C2. High skills and knowledge demands. DevOps practices require that, in the least, the team as a whole has all necessary skills and knowledge to develop, integrate, test and deploy, which includes provisioning and upkeep target environments and infrastructure. A well-established deployment pipeline helps new members to get up to speed fast, but eventually, they need to learn on how to update the deployment pipeline, which was perceived to require a distinct set of skills. In Case D, the transition to DevOps was considered especially demanding, as many new technologies and platforms were in use at the same time. In Case E, the team was considered on the cutting edge in terms of development skills and knowledge, but even they struggled with the learning curve of the DevOps approach. In addition to the requirement to be able to do everything, from low-level optimisation to deployment scripts, they were responsible for developing multiple services with different technologies and platforms. In Case B, DevOps was considered to demand much expertise from the team and to be risky if, for example, information security problems were created due to a lack of expertise. In Case A, the time and effort new developers required to get up to speed was considered a challenge, especially when the existing experts were unavailable to provide guidance due to other tasks.

> 'Being responsible for everything from even low level bug fixing, optimization all the way to the service, deploy to cloud wherever, it's a very large field to grasp. There are some things where I have to rely on my team members to remember how to do certain things or how some things have been built because even though we try to document what we do, it's never perfect' (Developer, Case E)

C3. Project and resource constraints. In Case A, the project was based on a fixed price contract with a pre-defined schedule and requirements. This meant that the team had to deliver all agreed-upon functionalities by the agreed-upon deadline. This not only delayed deployment to production until the end of the project, but also left little time for improving test automation and piloting production deployments, which in turn affected the quality of the system. Furthermore, any changes to the requirements of the project required a re-negotiation of the official project scope, which was frustrating for the team. In Case D, the development and operations teams were relatively small in number compared to their workload. For the developers, it meant much context switching between the 20 APIs under development. For the customers, the development was too slow. The operations team suffered from the lack of time to offer their support to developers, as they were also serving other teams. In Case C, development resource constraints meant that even important work was sometimes buried in the backlog due to a constant stream of new, even more important work.

C4. Difficulties in monitoring. In Case B, the development team did not monitor the containers in the containerised service. They could only monitor the servers and the overall availability of the service, but had no visibility of the internal health of the containerised microservices. In a similar vein, they did not have a centralised log of the containerised microservices. Instead, the logs of the containers need to be viewed one by one. Moreover, in Cases D and E, it was perceived to be quite difficult to know well in advance what metrics to monitor. Often, new monitoring metrics were introduced after the team experienced severe breaks in production (Case D).

C5. Balancing between speed and quality. In Case C, external deadlines sometimes forced the team to publish updates that had quality issues. Determining the right level of quality was observed to be challenging for the developers. In Case D, some customers were unhappy with the speed of the development team. The reason for the seemingly slow development speed was the high refactoring load. The development team had to spend almost half of their time on refactoring tasks. In Case A, the customer pressured the development team to implement new features instead of fixing bugs in the existing features. Due to the lack of emphasis on quality, the same bugs sometimes needed to be fixed multiple times in different features and the overall quality of the system was considered low. In Cases C and D, the development team strove to create a high-quality system, but the product owners preferred functionality to high quality. This sometimes caused some dissatisfaction on

Table 6: Perceived challenges in DevOps

Findings of perceived challenges from this study	Findings from literature
C1. Insufficiencies in infrastructure automation	Difficulties in automation of deployment process [22], [45]
C2. High demand for skills and knowledge	Steep learning curve for operations personnel and developers [34] [28] [15] [17]
C3. Project and resource constraints	**C6.** Difficulties in convincing senior management about DevOps [23],[17]
C4. Difficulties in monitoring, especially for microservice-based applications and in determining useful metrics	**C7.** Lack of common understanding of DevOps concept [14]
C5. Difficulties in determining a right balance between the speed of new functionality and quality	**C8.** Blurred responsibilities between development and operations [15]

the teams, as they did not like to produce low-quality software because of speed.

6. Discussion

This sections discusses the findings of the study in light of existing literature. Implications to practice, validity and limitations of the study are also discussed.

Our findings show toolchain use and support for the activities of the deployment pipeline in all cases, while also applying the principles and practices of trunk-based development, code reviews, CI and Agile and lean. Prior work [49] observed tool usage in software delivery to be affected by the state of current workflows, manual work and relevancy of tools. This study makes a similar observation, but, in addition observed that tool acquaintance by software development and operations teams limited usage because it required significant effort, time and investment. Steep learning curve, particularly among developers for operations tasks, was identified as the most common challenge perceived by all cases and is reported in prior work [15].

Automation of the deployment process is a common DevOps practice in literature [45]. Mäkinen et al [49] inquired for possible explanations to the observed great dierence between companies¢¢ capability to deliver and the actual release cycle. This study suggests two explanations that may apply in small and medium sized companies. First reason is the practice of having product owner manually test and approve new software features prior to deployment in production. Second reason relates to the selected deployment strategy, such as canary release that exposes software changes incrementally to a portion of users before deploying them to entire user base. As noted by [38],], certain deployment approaches used with the tools pose some reliability concerns. Their comparison of heavily baked image and lightly baked image deployment approaches showed that typically, heavily baked image approaches cause delays in deployment, even for minor changes [38]. Although we did not explore the implication of deployment approaches in detail, it was observed in studied cases that image baking took a long time (15 minutes in Case E) even for critical bugs could not be xed in less time. Studies, such as [39] [38] evaluate in detail dierent deployment approaches and reliability issues imposed by tools.

This study observed that it can be difficult to fully automate the deployment process due to context factors, such as the existence of legacy technologies and faulty network with reliability issues. The decision to move to an externally provided cloud infrastructure in studied case was seen in a positive light to tackle the issues. The negative aspect of the public cloud is that it imposes some technological and architectural restrictions to the system, which according to Cito et. al., [37] can be seen as a positive thing by developers in enforcing best practices and putting more focus on delivering customer value. Similar to our study, cloud infrastructure to a large extent is a major enabler of DevOps implementation; however, the use of public cloud remains challenging [22] and require acceptance of the client (for consulting companies) or regulations (for public entities).

Based on our study results, we build on our previous [30] enhancement of DevOps definition originally proposed by [32]: *"DevOps is a mind-set change* ***substantiated with a set of automated practices*** *to encourage cross-functional collaboration between teams–especially development and IT operations–within a software development organisation, in order to operate resilient systems and accelerate the delivery of changes. "*

Our analysis identied the following practical implications of DevOps adoption and implementation. First, as steep learning curve among software developers and operations stas is to be expected in DevOps implementation. Companies can facilitate and motivate learning through the practice and culture of knowledge sharing across different. Secondly, the responsibility of product owner to manually test and approve new software features before deployment in production signicantly affects the delivery speed. At the same time, the product owners, who have other parallel ongoing tasks, want adjustment to this responsibility. Therefore, it is important that value-adding activities be identied to adjust the responsibilities of product owners while also ensuring trust to development team and investments in automated test.

6.1. Validity and limitation of the study

The study acknowledges several threats to validity, against which some mitigation strategies were employed. Three cate-

gories of validity threats are discussed: construct validity, external validity and reliability [51] [52].

Construct validity. Construct validity is concerned with having researchers measure what was intended to be measured. In this study, threats to construct validity are related to whether the selected cases actually apply DevOps and whether the interview questions were interpreted in the same way by all interviewees and interviewers. Threats to the selection of the cases were mitigated by clearly specifying the goals of the study jointly with company representatives, who then selected appropriate cases based on the understanding of their organisations. In addition, having at least 10 researchers develop and review each question in the interview guide, as well as having pilot interviews to share and improve on the guide helped to equip the researchers to adapt to the questions, depending on the role of interviewee.

External validity. External validity is used to define the domain to which the study ndings are applicable and thus generalizable. Threats to external validity in this study originate from the participating cases being from one domain, i.e., web applications and services, and small and medium company size. Furthermore, some experiences may not be observed in large companies and system domain due to large system scale, characteristics and other context factors. Our ndings are not meant to be generalized rather provide insights into DevOps practices and their perceived benets and challenges requiring practitioners to evaluate their applicability in their contexts.

Reliability. The reliability of the study findings is concerned with the operability of the study design, such that it can be repeated with the same results. Researcher bias was minimised by involving at least six researchers in all data collection and analysis steps. Furthermore, the respective companies were involved to allow them to give feedback in different phases of the research.

7. Conclusion and Future work

This paper has presented a multiple-case study of DevOps implementation in five different software development contexts. As main contributions, the paper provides an enhanced definition of DevOps that gives emphasis to automation practices in addition to collaboration between software development and operations. Additionally, the detailed description of DevOps adoption and implementation presented in this paper have shown that DevOps provides ability, ownership and responsibility of software development teams to deploy software in production environment fast and often particularly for small and medium companies. In this context the capability comes with their ability to acquire new skills of operations-related tasks while at the same time cope with working under pressure. We identified that DevOps implementation in companies is a long-term activity that requires a supportive culture and mind-set in addition to the technical practices. Such cross-function collaboration is most effective when supported by senior management and customer.

We propose the following research areas based on our study findings:

- *Validation practices for deployment scripts* Complex infrastructures that involve manual steps were observed to cause reliability issues. Deployment scripts are at times erroneous, and there were no other means to validate them other than developers monitoring during deployment. Future research should consolidate knowledge of other possible technologies or practices for validating deployment scripts.
- *Balancing between speed and internal code quality (technical debt) in a commercial context* DevOps requires a comprehensive functional testing pipeline, but internal code quality might be deprioritised in favour of development speed. Because functional testing protects the outward visible quality, a low internal quality may not cause visible issues for a long time, but could become problematic at some point. Future research should investigate refactoring practices and the trade-offs in the balance between quality and speed in a non-open source context.

Acknowledgement

This work was supported by TEKES as a part of the N4S project of DIMECC (Digital, Internet, Materials and Engineering Co-Creation). We would like to also acknowledge and thank the companies and company representatives for their contributions and involvement in the study.

Appendix A: Interview guide

Introduction

- Role and daily responsibilities
- Years of experience

Management overview of the development process

- How are development and operations activities organised?
- How many teams do you have? Sizes of the teams?
- How are responsibilities given/shared to the teams?
- How would you describe <the case >, is it typical or specific in the organization?
- What kind of development process do you use?
- How are requirements and features specified, by whom?
- How are prioritization and decisions made?
- How are the features released to production, how often?
- How are the release decisions made? By whom?
- When and why did you start to implement DevOps?
- Who was/is the driver of DevOps implementation?
- What does DevOps mean in your organization?
- What kind of impacts do you see so far? Any hard data?
- What has been challenging in DevOps?
- How far along are you now in DevOps?
- What advice would you give for DevOps implementation?
- Decribe the case in detail e.g., product, teams, DevOps use

Development

- How is the project team organised?

- Who do you interact with the most? When? How?
- How are product requirements determined and incorporated in product development process?
- How often do you release the software and for what types of software changes?
- Can you describe the work-flows/practices until deployment and operational use in production
- What tools are used to support development work?
- When do you consider an implementation to be done?

Integration and deployment

- Describe in detail at which stages and levels software changes are integrated, built and tested?
- What tools are used to support building and testing?
- What practices, other than testing, contribute to quality?
- What is the process of deploying/delivering to customers?
- How is quality assured in the deployment pipeline?
- What are the most important challenges regarding quality?

Infrastructure management and monitoring

- Who (what team) is responsible for managing and maintaining infrastructure?
- Does development team have access to resources in production?
- How are non-functional requirements and information about configurations communicated to developers?
- How do you provision and manage different environments of your application?
- How are software changes deployed in production?
- What monitoring is performed in production?

Perceived impacts of DevOps

- What does DevOps mean to you?
- What impacts have you seen after introducing DevOps? Do you have any data to support this?
- Considering current practices in this team, How do they differ compared to before introducing DevOps?
- What have been the main benefits of DevOps?
- Are there any drawbacks in using DevOps?
- What are the main challenges of using DevOps?

Culture and mind-set

- Is your company culture supporting the DevOps approach?
- How would you describe your development culture and mind-set at organisation and team?
- How is management support for the development practices and culture visible?

Declaration of Competing Interest

All authors have participated in (a) conception and design, or analysis and interpretation of the data; (b) drafting the article or revising it critically for important intellectual content; and (c) approval of the final version.

- This manuscript has not been submitted to, nor is under review at, another journal or other publishing venue.

- The authors have no affiliation with any organization with a direct or indirect financial interest in the subject matter discussed in the manuscript

References

References

[1] F. Erich, C. Amrit, M. Daneva, A Mapping Study on Cooperation between Information System Development and Operations, in: International Conference on Product-Focused Software Process Improvement, Springer, 2014, pp. 277–280. doi:10.1007/978-3-319-13835-0_21.

[2] J. Humble, D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, 1st Edition, Addison-Wesley Professional, Boston, 2010.

[3] P. Rodríguez, A. Haghighatkhah, L. E. Lwakatare, S. Teppola, T. Suomalainen, J. Eskeli, T. Karvonen, P. Kuvaja, J. M. Verner, M. Oivo, Continuous Deployment of Software Intensive Products and Services: A Systematic Mapping Study, Journal of Systems and Software 123 (2017) 265–291.

[4] J. Humble, J. Molesky, Why enterprises must adopt DevOps to enable continuous delivery, Cutter IT Journal 24 (8) (2011) 6–12.

[5] B. Tessem, J. Iden, Cooperation between developers and operations in software engineering projects, in: Proceedings of the International Workshop on Cooperative and Human Aspects of Software Engineering, ACM, 2008, pp. 105–108. doi:10.1145/1370114.1370141.

[6] J. Iden, B. Tessem, T. Päivärinta, Problems in the interplay of development and IT operations in system development projects: A Delphi study of Norwegian IT experts, Information and Software Technology 53 (4) (2011) 394–406.

[7] J. Hamilton, On Designing and Deploying Internet-Scale Services, in: Proceedings of the 21st Large Installation System Administration Conference, USENIX Association, Dallas, Texas, 2007, pp. 231–242.

[8] P. Abrahamsson, J. Warsta, M. Siponen, J. Ronkainen, New directions on agile methods: a comparative analysis, in: 25th International Conference on Software Engineering, 2003. Proceedings., IEEE, 2003, pp. 244–254. doi:10.1109/ICSE.2003.1201204.

[9] O. Gotel, D. Leip, Agile Software Development Meets Corporate Deployment Procedures: Stretching the Agile Envelope, in: Agile Processes in Software Engineering and Extreme Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 24–27.

[10] P. Debois, Agile Infrastructure and Operations: How Infra-gile are You?, in: Agile 2008 Conference, IEEE, 2008, pp. 202–207. doi:10.1109/Agile.2008.42.

[11] D. E. Strode, S. L. Huff, B. Hope, S. Link, Coordination in co-located agile software development projects, Journal of Systems and Software 85 (6) (2012) 1222 – 1238.

[12] G. van Waardenburg, H. van Vliet, When agile meets the enterprise, Information and Software Technology 55 (12) (2013) 2154 – 2171.

[13] L. Bass, I. Weber, L. Zhu, DevOps: A Software Architect's Perspective, Addison-Wesley Professional, 2015.

[14] F. Elberzhager, T. Arif, M. Naab, I. Süß, S. Koban, From agile development to devops: Going towards faster releases at high quality – experiences from an industrial context, in: 9th International Conference on Software Quality, Springer, 2017, pp. 33–44. doi:10.1007/978-3-319-49421-0_3.

[15] K. Nybom, J. Smeds, I. Porres, On the impact of mixing responsibilities between devs and ops, in: 17th International Conference on Agile Processes in Software Engineering, and Extreme Programming (XP) , Proceedings, Springer, 2016, pp. 131–143. doi:10.1007/978-3-319-33515-5_11.

[16] A. Balalaie, A. Heydarnoori, P. Jamshidi, Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture, IEEE Software 33 (3) (2016) 42–52. doi:10.1109/MS.2016.64.

[17] S. Jones, J. Noppen, F. Lettice, Management challenges for DevOps adoption within UK SMEs, in: Proceedings of the 2nd International Workshop on Quality-Aware DevOps, ACM Press, 2016, pp. 7–11. doi:10.1145/2945408.2945410.

[18] P. Clarke, R. V. OConnor, The situational factors that affect the software development process: Towards a comprehensive reference framework, Information and Software Technology 54 (5) (2012) 433 – 447.

[19] N. Kerzazi, B. Adams, Who Needs Release and DevOps Engineers, and Why?, in: International Workshop on Continuous Software Evolution and Delivery, ACM Press, 2016, pp. 77–83.

[20] R. Jabbari, N. bin Ali, K. Petersen, B. Tanveer, What is DevOps?, in: Proceedings of the Scientific Workshop Proceedings of XP 2016, ACM Press, 2016, pp. 1–11. doi:10.1145/2962695.2962707.

[21] M. Leppanen, S. Makinen, M. Pagels, V.-P. Eloranta, J. Itkonen, M. V. Mantyla, T. Mannisto, The Highways and Country Roads to Continuous Deployment, IEEE Software 32 (2) (2015) 64–72. doi:10.1109/MS.2015.50.

[22] L. Chen, Continuous Delivery: Overcoming Adoption Challenges, Journal of Systems and Software.

[23] G. G. Claps, R. Berntsson Svensson, A. Aurum, On the Journey to Continuous Deployment: Technical and Social Challenges Along the Way, Information and Software Technology 57 (2015) 21–31. doi:10.1016/j.infsof.2014.07.009.

[24] PuppetLabs, 2016 State of DevOps Report, Tech. rep. (2016).
URL https://puppet.com/resources/whitepaper/2016-state-of-devops-report

[25] J. Davis, R. Daniels, Effective DevOps: building a culture of collaboration, affinity, and tooling at scale, " O'Reilly Media, Inc.", 2016.

[26] F. M. A. Erich, C. Amrit, M. Daneva, A qualitative study of devops usage in practice, Journal of Software: Evolution and Process 29 (6) (2017) e1885–n/a, e1885 smr.1885. doi:10.1002/smr.1885.

[27] B. Adams, S. McIntosh, Modern Release Engineering in a Nutshell – Why Researchers Should Care, in: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), IEEE, 2016, pp. 78–90.

[28] D. Cukier, DevOps patterns to scale web applications using cloud services, in: In Proceedings of the 2013 conference on Systems, programming, & applications: software for humanity, 2013, pp. 143–152.

[29] M. V. Mäntylä, B. Adams, F. Khomh, E. Engström, K. Petersen, On rapid releases and software testing: a case study and a semi-systematic literature review, Empirical Software Engineering 20 (5) (2015) 1384–1425. doi:10.1007/s10664-014-9338-4.

[30] L. E. Lwakatare, P. Kuvaja, M. Oivo, An Exploratory Study of DevOps: Extending the Dimensions of DevOps with Practices, in: The Eleventh International Conference on Software Engineering Advances, IARIA, Rome, 2016, pp. 91–99.

[31] B. B. N. de França, H. Jeronimo, G. H. Travassos, Characterizing DevOps by Hearing Multiple Voices, in: Proceedings of the 30th Brazilian Symposium on Software Engineering, ACM Press, 2016, pp. 53–62. doi:10.1145/2973839.2973845.

[32] R. Penners, A. Dyck, Release Engineering vs. DevOps-An Approach to Define Both Terms, Full-scale Software Engineering.
URL https://www2.swc.rwth-aachen.de/docs/teaching/seminar2015/FsSE2015papers.pdf{#}page=53

[33] L. Bass, R. Jeffery, H. Wada, I. Weber, L. Zhu, Eliciting operations requirements for applications, in: 1st International Workshop on Release Engineering, IEEE, 2013, pp. 5–8.

[34] J. Smeds, K. Nybom, I. Porres, DevOps: A Definition and Perceived Adoption Impediments, in: 16th International Conference on Agile Software Development, Springer, Helsinki, 2015, pp. 166–177.

[35] D. Stahl, T. Martensson, 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, 2017, pp. 440–448. doi:10.1109/SEAA.2017.8114695.

[36] Z. Babar, A. Lapouchnian, E. Yu, Modeling devops deployment choices using process architecture design dimensions, in: 8th IFIP Working Conference on The Practice of Enterprise Modeling, Springer, 2015, pp. 322–337. doi:10.1007/978-3-319-25897-3_21.

[37] J. Cito, P. Leitner, T. Fritz, H. C. Gall, The making of cloud applications: an empirical study on software development for the cloud, in: 10th Joint Meeting on Foundations of Software Engineering, ACM Press, 2015, pp. 393–403.

[38] L. Zhu, D. Xu, A. B. Tran, X. Xu, L. Bass, I. Weber, S. Dwarakanathan, Achieving Reliable High-Frequency Releases in Cloud Environments, IEEE Software 32 (2) (2015) 73–80. doi:10.1109/MS.2015.23.

[39] W. Hummer, F. Rosenberg, F. Oliveira, T. Eilam, Testing idempotence for infrastructure as code, Middleware 2013.

[40] J. Wettinger, V. Andrikopoulos, F. Leymann, Automated Capturing and Systematic Usage of DevOps Knowledge for Cloud Applications, in: IEEE International Conference on Cloud Engineering, IEEE, 2015, pp. 60–65. doi:10.1109/IC2E.2015.23.

[41] T. Schlossnagle, Monitoring in a devops world, Communications of the ACM 61 (3) (2018) 58–61. doi:10.1145/3168505.

[42] C. Vassallo, F. Zampetti, D. Romano, M. Beller, A. Panichella, M. D. Penta, A. Zaidman, Continuous delivery practices in a large financial organization, in: IEEE International Conference on Software Maintenance and Evolution, 2016, pp. 519–528.

[43] L. Bass, R. Holz, P. Rimba, A. B. Tran, L. Zhu, Securing a Deployment Pipeline, in: 2015 IEEE/ACM 3rd International Workshop on Release Engineering, IEEE, 2015, pp. 4–7. doi:10.1109/RELENG.2015.11.

[44] A. A. Ur Rahman, L. Williams, Software security in devops: Synthesizing practitioners' perceptions and practices, in: Proceedings of the International Workshop on Continuous Software Evolution and Delivery, ACM, 2016, pp. 70–76.

[45] M. Callanan, A. Spillane, DevOps: Making It Easy to Do the Right Thing, IEEE Software 33 (3) (2016) 53–59. doi:10.1109/MS.2016.66.

[46] M. Fazal-Baqaie, B. Güldali, S. Oberthür, Towards DevOps in Multi-provider Projects, in: 2nd Workshop on Continuous Software Engineering, Vol. 1806, CEUR-WS.org, Hannover, 2017, pp. 18–21.
URL http://ceur-ws.org/Vol-1806/paper03.pdf

[47] T. Schneider, Achieving Cloud Scalability with Microservices and DevOps in the Connected Car Domain, in: CEUR workshop proceedings on continuous software engineering, CEUR-WS.org, 2016, pp. 138–141.
URL http://ceur-ws.org/Vol-1559/

[48] C. Tang, T. Kooburat, P. Venkatachalam, A. Chander, Z. Wen, A. Narayanan, P. Dowell, R. Karl, Holistic configuration management at Facebook, in: Proceedings of the 25th Symposium on Operating Systems Principles, ACM Press, 2015, pp. 328–343. doi:10.1145/2815400.2815401.

[49] S. Mäkinen, M. Leppänen, T. Kilamo, A.-L. Mattila, E. Laukkanen, M. Pagels, T. Männistö, Improving the delivery cycle: A multiple-case study of the toolchains in Finnish software intensive enterprises, Information and Software Technology 80 (2016) 175–194. doi:10.1016/j.infsof.2016.09.001.

[50] Y. Liu, C. Li, W. Liu, Integrated solution for timely delivery of customer change requests: A case study of using devops approach, International Journal of U-and E-Service Science and Technology 7 (2) (2014) 41–50.

[51] R. K. Yin, Case Study Research: Design and Methods, 5th Edition, SAGE Publications, 2013.

[52] P. Runeson, M. Höst, Guidelines for Conducting and Reporting Case Study Research in Software Engineering, Empirical Software Engineering 14 (2) (2008) 131–164. doi:10.1007/s10664-008-9102-8.

[53] K. J. Stol, P. Ralph, B. Fitzgerald, Grounded theory in software engineering research: A critical review and guidelines, in: 2016 IEEE/ACM 38th International Conference on Software Engineering, 2016, pp. 120–131.

[54] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer Science & Business Media, 2012.

[55] L. E. Lwakatare, T. Karvonen, T. Sauvola, P. Kuvaja, H. H. Olsson, J. Bosch, M. Oivo, Towards DevOps in the Embedded Systems Domain: Why is It So Hard?, in: 49th Hawaii International Conference on System Sciences, IEEE, 2016, pp. 5437–5446.

[56] V. Braun, V. Clarke, Using thematic analysis in psychology, Qualitative Research in Psychology 3 (2) (2006) 77–101. doi:10.1191/1478088706qp063oa.

[57] M. Shahin, M. A. Babar, L. Zhu, The Intersection of Continuous Deployment and Architecting Process, in: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ACM Press, 2016, pp. 1–10. doi:10.1145/2961111.2962587.