Date of acceptance Grade

Instructor

Teaching Container-based DevOps Practices in Higher Education Context

Jami Kousa

Helsinki February 28, 2020

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

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

Tekijä — Författare — Author
Jami Kousa

Työn nimi — Arbetets titel — Title
Teaching Container-based DevOps Practices
in Higher Education Context

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
	February 28, 2020	47 pages + 0 appendix pages

Tiivistelmä — Referat — Abstract

Teaching DevOps is challenging as it is inherently cross-functional between development and operations. This thesis presents an examination of a course and its development iterations in an attempt to find an approach that may be used in DevOps education in a higher education context. The course focuses on learning a tool in the DevOps toolchain, containerization with Docker, in a massively open online course (MOOC). By investigating a course during its design process and its attendees, challenges that students and course instructors faced are discussed. The primary source of information from students is a survey that students answered before and after participating in the course. The challenges of teaching DevOps practices vary from the teaching methods to types of exercises and level of industry imitation or abstraction. In comparison, students had fewer challenges than expected with the course contents. The survey results offered insight into the student experience concerning DevOps, unveiling a demand for both further development of the course as well as for new courses that link development and operations.

ACM Computing Classification System (CCS):
Applied computing → Education → E-learning
Software and its engineering → Software notations and tools → Software configuration management and version control systems

Avainsanat — Nyckelord — Keywords
devops, teaching, university

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction

Software engineering consists of multiple different parts. Analysis of requirements, design, coding, verification, and testing are the areas of software development. In contrast, maintenance and installation are considered to be the responsibility of software operation [1]. In total, three parts of the software lifecycle are classified under software operation; deployment, operation, and maintenance. DevOps is a methodology that is used to describe the joining of development and operations work and the management of software delivery with multiple methods such as using automated delivery pipelines [2].

Most university students are taught software development with their studies in computer science. As such, they have a solid understanding of the ecosystems used in creating software. However, even though the most recent ACM Curricula recommendation for computer science also includes deployment, operation, and maintenance as part of software engineering [3], the operations part often receives little focus. This lack of attention is visible also in the ACM Curricula recommendation: course examples in the recommendation seem to be missing deployment, operation, and maintenance of software.

This problem has been identified at the University of Helsinki as well. The curriculum at the University of Helsinki contains software engineering education for aspiring developers. Nevertheless, learning the operations side has been mostly left as a personal responsibility of the student, who is expected to either attend optional courses or learn about the topic on their own. Such a situation where relevant toolchains are poorly included in the curriculum has also been reported elsewhere [4].

Aiming to bridge the gap between development and operations, DevOps brings the three missing practices from ACM curricula closer to development. Integrating DevOps education into the university curricula is seen as a potential method of introducing the operations work to students. Teaching DevOps is challenging as it is crossing between boundaries of traditional computer science courses offered at universities, and requiring both development and operations expertise [5]. These practices that DevOps covers may also be challenging to learn for students in the constraints of a course.

1.1 Research questions

The research questions emerged from the challenges of teaching DevOps practices. As a part of this work, a new course was created that focuses on containerization with a tool called Docker. Also, as part of the course design process, the course was iterated upon to improve on areas in which students have difficulties. Finally, an inspection into the course and its attendees via various methods was conducted to solve the research questions. Overall, this process highlighted the possible challenges of implementing a compulsory course into the curricula.

- RQ0: How are skills related to DevOps taught now?
- RQ1: What are the prerequisites to start learning containerization as a DevOps practice in university?
- RQ2: What kinds of exercises can be offered to students to enforce learning DevOps practices?
- RQ3: What are the core challenges for students when learning DevOps practices?

To answer RQ0 the current state of education related to DevOps at the University of Helsinki was analyzed. The lack of operations education has been identified in a previous analysis of the curriculum [4].

To answer RQ1 data was gathered from the course attendees before, during, and after the course via various methods. These include personal communication from attendees of the first instances and two surveys during the final instance of the course. During the initial design phase of the course, prerequisites were constructed from personal experience of the technologies involved and the current curriculum.

For the RQ2 and for the design phase of the course, it was essential to look into methods used in teaching in general. Instructional scaffolding is used as the primary method; students are first instructed to do simple tasks until they are capable of applying the learned skills to solve more complex problems [6]. During the course, the student feedback was taken into consideration. The course was iterated upon to reach conclusions on the types of exercises that were better for the students and their learning outcomes.

For the RQ3 students were given a chance to give feedback during and after the course. A survey at the end of the course encouraged attendees to describe which

exercises were the hardest for them. The most challenging exercises and subjects were revealed by the the survey answers. As the course had multiple instances, there were also opportunities to interview a small subset of students that participated in workshops to gain help with the exercises. These students' feedback was used to iterate the course between instances hoping to improve on the existing examples and material.

1.2 Thesis structure

The next chapter will present a review of the contents and a definition of DevOps. This review is done to inspect the possible methods of teaching DevOps via the DevOps practices and to narrow it into a course format. The third chapter will focus on the overall teaching of DevOps by starting with an analysis of the existing education at the University of Helsinki as a form of initial problem analysis to answer RQ0. The third chapter also depicts a new course teaching DevOps tools, that was designed as part of the thesis progress and released in the University of Helsinki. The fourth chapter introduces the methods used to answer the rest of the research questions and the design iteration of the course. After which the fifth chapter presents the finalized results. The sixth chapter contains discussion based on the results, and in the last chapter are the conclusions.

2 Software engineering and DevOps

Software engineering is a discipline that encompasses all phases of the lifecycle of a software system, including requirements elicitation, analysis, and specification; design; construction; verification and validation; deployment; and operation and maintenance [3]. The phases split into two different groups: software development and software operations. Software operations encompass the later phases of the software lifecycle after development [1].

DevOps has numerous definitions, and its definers have different views that have led to a meta-analysis on the definition. A systematic mapping study by Jabbari et al. offers a comprehensive definition for purposes of introduction [2]. This definition is inspected and reviewed for the use of it in the context of this thesis.

Jabbari et al. defined DevOps as follows:

> DevOps is a development methodology aimed at bridging the gap between Development and Operations, emphasizing communication and collaboration, continuous integration, quality assurance, and delivery with automated deployment utilizing a set of development practices.

The definition describes DevOps as a development methodology. As a development methodology, it includes different methods and practices for its practitioners to reach their goals. DevOps can be compared to agile as they are both development methodologies. While DevOps does not meet all of the principles of agile, DevOps can be considered to be an extension of agile [2].

The definition chosen, however, leaves open the methods and attributes that are used to bring Development and Operations together. Commonly it is thought that this means the streamlining of operations work by introducing practices such as infrastructure as code and continuous deployment and, with this, bringing operations closer to developers. Operations work may include other maintaining tasks, some of which are avoided with PaaS (Platform as a Service) or IaaS (Infrastructure as a Service) cloud services, both of which move the hardware maintenance to be the responsibility of another party.

A singular tool or practice may also cover multiple parts of the definition offered by Jabbari et al. For example, continuous integration and automated deployment are both often achieved with, or as a part of, deployment pipelines. The definition

leaves the implementation up to the practitioner of DevOps. DevOps engineers use a diverse set of tooling to resolve the challenges in the implementation of DevOps. It is also worth noting that quality assurance is added to the definition by Jabbari et al. despite there being only a relatively small number of definitions saying that DevOps concerns quality assurance [2]. However, the targets are overlapping again within the tools: automated deployment provides the possibility to respond at a much faster pace to bugs in software [7]. A fix that takes minutes to program may also be deployed without further work, thus leading to the *continuous* part of delivery or deployment.

Communication and collaboration in the definition is in the context of between developers and operations [2]. An example of an anti-DevOps pattern to contrast the collaboration would be that developers develop something that works on their machine and then push the product forward to the operations team, which in turn has to start working on the product.

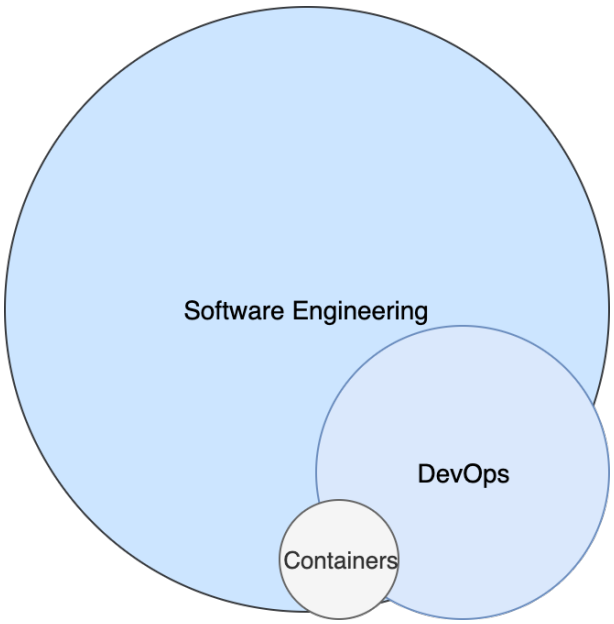

Figure 1: DevOps and its tools extend beyond Software Engineering

The Venn diagram in Figure 1 illustrates DevOps being part of the discipline of software engineering. In Figure 1 DevOps also extends outside software engineering as the whole IT operations is the concern of DevOps, hardware operations included [8]. Figure 1 also presents a tool, containers, that are used in some DevOps practices and illustrates that the containerization and it's use cases are not limited to DevOps or even software engineering.

2.1 DevOps toolchain

The main effort for DevOps to succeed is to make the overall process of software delivery continuous [9]. The DevOps toolchain consists of several different processes and tools that assist in automation and adoption of the continuous processes.

Continuous deployment is the term used to describe an approach to software engineering in which the software is automatically deployed as the software is developed. There are various methods to do continuous deployment, but all of them require a deployment pipeline. A deployment pipeline is an automated system that takes the code that developers create and delivers it to the user. There are various methods of implementing a deployment pipeline. An example of a deployment pipeline could consist of three parts. First the code is version controlled by the developers. The second would test the code and compile it into an application if the tests passed. And as a final step the application would be deployed to a remote server after it has been compiled. Configuration management may be used to manage the configuration within the deployment pipelines.

Configuration management is a discipline for controlling the evolution of software systems [10]. There is a large number of tools created for configuration management. The tools range from version control systems for code, such as git [11], to operating system configuration management aiding in the management of configurations and versions of operating systems and applications in them, such as Ansible [12] or Puppet [13].

Infrastructure as code is a concept used to describe servers that are defined with configuration files. Many modern configuration management tools use infrastructure as code to establish a declarative method to define infrastructure. Infrastructure as code is considered to be a part of the DevOps toolchain as a major DevOps enabler [14]. Containers, or operating system level virtualization [15], can be used to make infrastructure as code easier as they are declarative method to define infrastructure. Containers couple the code with the environment that it requires, often so that the containerized environment is version controlled with the code of the software.

Immutable infrastructure is another method to deal with system configuration management. The core concept of immutable infrastructure is that the state of the infrastructure is not changed. Instead of changes, to modify the existing infrastructure in a system, a completely new version replaces the old [16]. As a result, the infrastructure is not dependant on previous changes and can be torn down and set up

without overhead. Ambiguous or hidden configurations in the production environment do not exist as everything can be set up with the instructions. In addition to this, as infrastructure is kept immutable from testing to production, differences between environments are avoided. In some cases, containers can provide immutability as the environment is configured unambiguously via build instruction for the initial contents of a container. The same configuration can then be used throughout the deployment pipeline from the programmers' machine to the production.

Immutable infrastructure is a significant shift from the more traditional world of configuration management where changes are applied as incremental updates to an existing system [16]. The conventional methods of running configuration management are either that the operations team is working as a bottleneck to all changes to production or developers run configuration management for what they are responsible for [17]. To avoid the issues with code management and out of sync configuration in the latter scenario, immutable infrastructure with containers can be seen as a solution [17].

2.2 Containers and Docker

Containers are operating system level virtualization designed to be a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another [15]. The benefits of containerization are similar to other virtualization technologies such as virtual machines while outperforming them with near native-performance of CPU, memory, disk, and network [18, 19, 20].

Containers facilitate the work of DevOps teams by allowing the use of code to perform infrastructure management tasks automatically [21]. Containers offer fast deployment and recovery, supporting continuous deployment [22]. Quality deliveries with short cycle time, the time for a development task to reach production, need a high degree of automation and tools are mandatory in automating DevOps [23].

Containers answer to many of the challenges of following DevOps methodology. Containers are often considered to be a silver bullet [24] to solve problems with dependencies. Also, as containers close the gap between developers' environment and the final deployment environment, the work of developers and operations is brought closer together whilst making the process more transparent. DevOps can enable continuous integration and deployment [25]. Containers can seemingly be

used to deploy applications without risk of versioning issues and promising additional speedups and a higher level of abstraction in the continuous integration or deployment pipeline [26].

The benefits of containerization are not simply in software engineering. As is illustrated in figure 1. Research can also benefit as it also has difficulties with the differences between environments, missing documentation, and changing dependencies [27]. By offering a platform for reproducible research, Docker skills may be considered to be a valuable asset for any computer science student at the university level. In computational sciences, this reproducible research may be considered as the minimum standard for assessing scientific claims [28].

Docker [29] is an open software for containerization. Docker is a DevOps tool that provides a systematic way to automate the faster deployment of Linux applications inside portable containers [30] and has been adopted in the industry as arguably the de facto -standard method for containerization [26].

2.3 Literature on teaching DevOps

One of the most comprehensive reports on teaching DevOps was published in 2016 by Christensen [5], who describes the challenges and proposes teaching methods for DevOps. Discussion in the paper on the lessons learned is done in relation to a course in which students developed web software using DevOps and cloud computing practices with the requirement that the software is able to scale to tens of thousands of concurrent users.

In a similar vein, Pengxiang and Leong [31] were the forerunners who published well-grounded research on teaching DevOps related hybrid-skills in the cloud computing environment. They analyzed the skillset wanted by the industry identifying a gap in the current polytechnic cloud computing curriculum. Their proposed implementation strategy for learning hybrid skills uses the software that students deploy as the platform for students to submit their reflection journals.

A recent publication by Kuusinen and Albertsen in [32] explores the university-company collaboration in teaching Continuous Delivery and DevOps. In their study, the students had ten full contact days, 125 hours of learning hours, over two calendar weeks. The first of the weeks was theoretical, and the second was practical, and the learning outcomes contained various competencies, not just skill-based competencies.

The work by Christensen, as well as Kuusinen and Albertsen, identify the scarcity of research done on teaching DevOps in higher education. The mentioned scarcity mostly concerns the methodology and culture of DevOps. Some DevOps practices are introduced in other courses to provide some experience of their usage. Version control management systems, such as git, which may be classified as DevOps tools, can be introduced in many courses as Haaranen and Lehtinen describe [33]. Other tools, such as continuous integration using CircleCI [34], are integrated into software engineering projects. These kinds of integrations make it possible for students to learn DevOps tools and practices during courses that do not focus on DevOps as a methodology.

3 The overall state of DevOps education

For software engineers and computer scientists joining the software development industry, a lack of configuration management experience has been considered to be a major, if not the greatest, knowledge deficiency [35, 36]. Configuration management discipline is central in all of the major processes of systems and software engineering [37]. The importance of configuration management and the lack thereof is a problem for computer science students as their studies do not prepare them for industry.

As DevOps is integrating multiple fields, a problem arises with the nature of University curricula. The topics have been mostly attempted to be shoehorned into existing courses [38]. Embedding these skills is not a trivial task, requiring a deeper understanding of the nature of the fields DevOps is integrating and requires faculty with sufficient experience in operations, which is not an easy task [39]. Existing courses include education on a select number of tools categorized as DevOps tools such as version control systems (git). These courses include software engineering projects which may offer a broad look into the software engineering discipline. However, as the industry moves forward, some of the practices that are taught in courses are potentially lagging behind [40].

DevOps imposes cultural changes to software development as well as to organizational structure. Only the tools side of DevOps is considered in the context of this thesis. This delimitation is to narrow the context, and since the main effort for DevOps to succeed is to make the overall process continuous [9] starting from the tools may be an excellent method for introduction. Also, the DevOps culture may be more efficient to learn in a group project. Containerization, more specifically Docker, was chosen as the subject for a new course. Docker can be considered to currently be the de facto -standard as the tool for containerization in the industry.

DevOps is not well defined even in the industry, and this has led to the problem that companies may not know what skills to look for [38]. The definition of Jabbari et al. introduced in previous chapter consists of 5 parts, but there is not a single universally agreed-upon definition. The context for the analysis of the current state is the University of Helsinki, Department of Computer Science. The offered definition is used as it is the most comprehensive definition available to make it possible to talk about the current state of DevOps education at the University of Helsinki.

3.1 Initial problem analysis at the University of Helsinki

The University of Helsinki has divided Master's level computer science studies into multiple tracks. The five parts of the definition offered by Jabbari et al. [2] are used to analyze the existing courses in the computer science curricula. The learning goals of the courses are considered as they offer a sufficient approximation to the contents of the course, as students are not expected to learn outside of the learning goals. Courses that are not part of the software engineering track are not included as they are not intended for software engineering students to participate in even as part of optional courses. Of 39 mandatory courses and 54 optional courses totaling in 93 courses are 11 courses satisfying the definition at least partially: 8 mandatory and 3 optional courses, 4 masters-level courses, and 7 bachelor's level.

Table 1: Courses available at University of Helsinki that contain topics in areas that DevOps is aimed at

DevOps consists of	**Number of courses teaching**
Bridging the gap between Development and Operations	2
Emphasizing communication and collaboration	6
Continuous integration	3
Quality assurance	8
Delivery with automated deployment utilizing a set of development practices.	1

An investigation into existing courses was made from personal experience and descriptions of the mandatory courses in computer science to answer RQ0. Most of the courses that appeared to contain DevOps methods were project courses. The only course that did contain every aspect of DevOps was the Software Engineering Lab -course. However, if the part "Emphasizing communication and collaboration" of the definition is interpreted to only be in the context of between different teams of development and operations, it can be considered to be missing from the course. Software Engineering Lab is considered to be the culmination course of a bachelor's degree with a customer project. There are no courses with project work with multiple teams working on the same project, except sometimes Software Engineering

Lab teams split into two development teams. The interaction between operations and development is not seen. The requirements for industry-level expertise in DevOps are hardly met with the current curricula. The requiring of further operations education corresponds to that observed by Luukkainen et al. previously [4].

Cloud services and web development can be considered to be early adopters of DevOps practices [23]. Both of them are taught in separate optional courses, and both topics could utilize containerization. Containerization could be integrated as a part of other courses, but a dedicated course offers the benefit of modularity for students. It opens the possibility of offering the course to anyone interested in containerization itself. In addition, as DevOps is a broad subject, it could also be followed up with an advanced course.

3.2 Course design

As DevOps spans a large part of software engineering rather than going over everything, the prerequisite for designing a course was to narrow down to its components. The tools in a DevOps toolchain were chosen as the course topic as they offer a possible entryway into DevOps mentality. Containerization using Docker was primarily chosen as the course subject because, in addition to its status in the industry, it was familiar for the course designers.

For purposes of teaching industry tools and their usage, a course "DevOps with Docker" was designed and held as an optional course in three separate instances starting November 2018 and ending January 2020 at the University of Helsinki. The course had 84 + 304 attendees in total. The goal was to teach students the fundamentals of Docker as a tool and to show the benefits of using containerization in various environments.

Containerization was chosen as the overarching solution to feature the problems and solutions used in DevOps. Containers themselves do not offer a complete picture of DevOps, lacking key subjects such as monitoring. Containers offer the means against a subset of the presented problems, such as "works on my machine." This divide between the development and production environments is a significant hurdle as the classic "works on my machine" divide opposes DevOps principles [41]. The teaching of methods to solve this problem offers students the possibility to design continuous deployment pipelines when confronted with such a task.

For the course format, an online course was chosen. The course material was written

initially for an industry level introduction to Docker [42]. The transfer from industry context to the university context meant the inclusion of exercises and a grading process. The online format made it possible to offer the course as part of open university and offer the course during unconventional periods such as during the winter break. The course was designed to work as a MOOC (Massive Open Online Course) without lectures or strict time constraints. The decision for an online course introduced another group of problems for the course and its development.

The tools that software engineers use have been taught to students as part of their standard course work. For example, version control systems such as git can be used as a platform during other courses to teach students its usage [33]. There is nothing entirely new with the course method or subject as other DevOps courses exist [5], but the online format is presenting challenges that would not be expected from a lectured course.

A MOOC course about DevOps with Docker faces the same challenges as other MOOC courses. With massive open online course comes the impossibility of providing assessment and feedback that is not either automated or peer assessed [43]. The course was designed so that the instructor could stay as an outsider support as the students progressed with the material to mitigate the issues brought by the online format. The investigated course can be classified as an asynchronous mini MOOC as it does not conform to the traditional timetable of semesters [44]. The short duration of the course and the lack of timetable on the course is also considered positive by students in general [45].

Teaching DevOps has multiple challenges varying from students being expected to have minimal experience as well as requiring operations expertise from teacher [39, 5]. One of the initial challenges that were brought up in the course design phase was that the benefits of containerization might seem abstract to students. Extensive descriptions of use cases may not be as efficient without hands-on experience on the subject. As such, an emphasis on exercises over reading material was chosen as the basis for the course.

Self-assessment can help develop the self–learning skills and improve the ability for self–analysis [43]. Most exercises were designed with clear success conditions with an assisting application as part of the exercises or other clear indicators for the successful completion. Besides, the exercise description also instructed how the expected result should look. The primary application that the students containerized had multiple buttons that turned from yellow to green on successful exercise completion.

The application made it possible for the exercises of the course to be completely self-assessed. When facing difficulties, students were encouraged to use any resources available to them outside of course material.

The course contains an introduction to containerization with Docker and container orchestration with docker-compose. With both tools, students are exposed to various use cases.

3.2.1 Course contents

The course consists of 4 parts. The first part, labeled as part 0 in course material, consists of installation instructions for the students to get started. Part 1, or the second part, is focused on Docker itself and its usages as a tool. Part 2 introduces the students to docker-compose, a tool that is used with Docker to run multi-container applications on a single host. Part 3 builds on previous parts and offers some best practices, optimization methods, and finally introduces other related technologies to students without in-depth content or exercises concerning the implementation or usage.

At the beginning of the course, we define the most important concepts to get started: definitions such as what is an image or a container are at the beginning. Rather than spending time explaining the structure of the docker environment/runtime, the course examples start right after with usage of the everyday functions of the Docker CLI and introduce the Docker daemon. The course focuses on using Docker as a problem-solving tool in contrast to a more theoretical approach to how operating system level virtualization works or is implemented.

An explanation of images and the building of one was presented after the introduction of Docker images via the Docker CLI commands. For this, the course has an example application that starts in part 1 and the final example with that application is in part 3. These examples used in the material use a command-line application. In addition the first part introduces students to the pre-programmed web application consisting of multiple parts with an increasing number of containers during the progression of the course. These exercises start by merely creating a Dockerfile to build an image.

After learning how to use Docker to containerize applications, the course moves to docker-compose, a tool to orchestrate multi-container applications. [46]. Introduction to docker-compose makes it possible to set up more complex applications in a

single host. With the exercises, students are also introduced to multiple parts of applications: databases, reverse proxies, caches, APIs. These were introduced with a single expanding a web application that they first containerized in part 1. The web application initially consisted of two containers; simple container with static assets served to a browser and a container with a REST API that was interacted with by the frontend in the browser. The second part with docker-compose also includes an example and an exercise with container scaling with an application increasing in performance while scaling.

Before reaching part 3, students had already learned how to use Docker in most situations with the CLI and docker-compose, respectively. The final part was created to introduce students to the more complex parts of Docker and where to go next with containers and orchestration. In this section, students familiarize themselves with multiple ways to minimize the size of an image and find out about other issues in their previous images and containers. The most important one of these is security as Docker, by default, offers root access. The end of part 3 also contained introduction to other tools of container management: what students had learned during the course may not be enough to deploy to production, for example, on multiple host computers.

The exercises are instructed to be completed at specific points in the material. Instruction to complete a particular exercise was given as soon as the concepts were taught through the material. With instructional scaffolding, many exercises differ or go beyond the examples and may require the student to find further information about the subject at hand. Reading the documentation was considered to be a part of the course as tools, and their usage may change. Learning to read documentation is a significant part of software engineers' work practices, with most software engineers considering reading documentation a daily practice [47].

Most of the exercises were either the implementation of containerization to a web application or a build-up exercises for the skills to a more complex implementation at a later exercise.

In the exercises, students configured different kinds of applications and architectures to work within a container or a container network. The pre-programmed applications were designed so that students could see the application working, and most of the problems faced by students were error messages. The messages provided an excellent chance for students to learn to read error messages and made sure the students reviewed their solutions before submission. The course did not use automatic

reviewing as students were expected to not cheat during the course; students were instructed that cheating would lead to consequences.

A form of instructional scaffolding is present in some of the later exercises as students were also encouraged to implement the same technologies and solutions in their personal projects. This was done via exercises that require the students to link their open source project and share what they had done. Students were instructed to submit working applications through GitHub.

3.2.2 Limitations of the course

Some of the cases where DevOps can be of use are challenging to reproduce, requiring multiple servers [5]. Such external limits can influence the implementation of the course regarding DevOps with it being skill-oriented. Realistic context is possible with the material, but as the examples in the material were simplified versions, the more realistic context was offered with the exercises. However, as a realistic context is vital with DevOps, it is critical to offer such with the exercises [5]. The course only considers Docker in a limited number of production environments, mainly single-host environments where connections between containers are relatively simple.

Several other issues should be noted about the course. The students are mostly expected to self organize their studies which can lead to differences between students that are not related to the technologies. Either because the student is not experienced in research that may be needed to complete some exercises or because students can plagiarize solutions from other students to get a passing grade since there is no course exam. Another problem is that the course has a device and operating system requirements. Students are required to have their administrator privileges on their computer, and notably, Windows Home edition is not sufficient to run the Hyper-V virtual machine supervisor which is required to run Docker on Windows.

For the problem regarding students self organizing their studies, we have offered a social media solution so students can ask each other and the instructor for advice. The almost 24/7 peer support makes it possible to help students struggling with exercises in real-time. Through an application, students could send instant messages to everyone else attending the course and get peer help with the exercises. University students could be trusted not to share solutions outright as it would make it moot to attend the course. To avoid possible exploitation of the course arrangements, a subset of the student solutions were manually checked.

Justification for not doing rigorous plagiarism checks is that the students are partially responsible for their studies as well. The benefits are small, with the risk of getting caught. The only benefits of cheating by copying solutions or delivering uncompleted exercises are the course credits of an optional course. The credits from an optional course can be considered to be less critical for a degree compared to mandatory courses.

Problems regarding required devices can be solved with virtual machines, although it requires more work by the student. The limitation is expressed at the start of the course so students could be fully prepared.

Limiting the total extent of the contents of the course to the usage of the Docker tool is sufficient to teach the benefits and usage of containerization, although there is no real blocker to extend the course for more high-level technologies intended to be used in environments with multiple host machines.

A significant remaining issue with the course is that many of the problems solved by DevOps may be somewhat abstract to students who may not have worked on a project of size and complexity [39]. Predicting the requirements to get started with Docker is not trivial. Students were not expected to have used or heard of containerization before joining the course.

4 Course iterations and the student survey

In this section the study setting is detailed with the iterations of the course. The course was continuously evolving the finalized form is presented in the last iteration. This iteration with student feedback and experience between course instances also reveled information related to the research questions.

Feedback from attending students was collected during and after three instances of the course, at the University of Helsinki. Of the three instances, two were in 2018 and one in 2019. The basis for course material was initially written for a course aimed at people working in the industry and then built upon to create an introductory course to containerization for students at the university level.

Prerequisites may present a challenge for students that do not have them. However, with the nature of DevOps, every student may have some missing requirements. Some course prerequisites can be introduced as a part of the course in the form of a reminder. The reminders were added to guide students that do not have experience with the subject to avoid getting stuck and finally master the subject. Targeting bachelor's students that may have minimal knowledge of configuration management and operations, in general, requires the course to have an introduction to most concepts that are not taught on mandatory courses.

There are two types of requirements regarding DevOps. First, are the skills from both the development and the operations sides that are required to start practicing DevOps methods. The second is the understanding or experience of the problems DevOps is designed to solve. The first set of skills can be taught via course material and exercises. The latter can be explained, but most of the problems that DevOps solves are not seen before entering the industry and moving projects out of development environments and into production. Monitoring as part of software maintenance is a solution that is often not addressed during development.

4.1 Survey

There were two surveys conducted during the course. First of which was offered to attendees at the beginning of the course. The latter was offered to attendees after each part as the attendee could end their completion there and receive credits or a certificate.

As the course started in spring and lasted until the next year, the number of com-

pletions was expected to be considerably lower than the number of attendees that would start the course. However, the number of attendees that completed the course in time for this study was even smaller. The attendees were not required to fill in the survey. A total of 499 attendees answered the survey at the beginning of the course. The survey at the end of the course received 67 answers.

The surveys were conducted on an online platform. The online platform that was used had a limitation of ten questions for each survey; as such, both of the surveys had exactly ten questions. The survey had the following questions using the Likert scale:

- I have a general understanding of containers
- I have understanding of the problems containers solve.
- I'm familiar with DevOps or have general understanding what it is

As well as on prior experience with the question: I have used Docker before / Before starting this course I had used Docker And in addition multiple-choice questions for further analysis:

- My goals for this course (were)
- My background is
- I have work experience of Docker or containers are used in my place of work
- I'm familiar with DevOps or have general understanding what it is
- I found this course via
- I'm looking forward to.

The small number of questions was considered to be useful as the risk of lower quality data is diminished in shorter surveys [48]. With limitations of the survey platform, arranging two studies was challenging, as many of the same questions had to be repeated at the end. As the survey was conducted with the total anonymity of the attendees linking the answers individually and verifying the attendees' completion and their survey answers was not considered. After the course, there was one additional question for the amount of parts attendees completed. Each part offered the attendees a single credit to a total of 3 credits. Overall, both of the surveys mostly contained the same questions.

4.2 Course iterations

The course has had three instances named alpha, beta, and open. Alpha and beta can be considered to be an in-house testing phase and were held right before and during winter 2018, respectively. The first two versions were only accessible by students at the University of Helsinki. The spring 2019 instance was released in open university. The course was hosted online on GitHub pages, and as such, during the course, students could send pull requests, change suggestions, to material with git to fix issues.

As the course was genuinely open, including the material and website, and included clear instructions about the pull requests, it was possible to improve the course continuously as the course was ongoing. This method has resulted in 35 pull requests from 21 contributors. This offered a chance for students to help their peers indirectly by improving the readability of the material. As parts of the material were changed during the course, the contents were kept up to date. Everything could not be updated as students had no time limits to complete the course. Removing or adding exercises would not be fair for students, and those changes were on purpose planned for after the end of an iteration.

4.2.1 Alpha instance

The first instance was a "closed" release with students personally invited to join the course, as opposed to a course available for sign up. The number of students who signed up was limited, and as such, the feedback methods were more personal. The feedback used to refactor existing parts of the course was collected mostly via personal communication with the students during the course.

The course had less than 20 attendees, of which two completed all of the exercises. The first iteration was still ongoing as the second iteration started, but the alpha students had an exclusive chance to review and complete exercises before the exercises were released for the next iteration. Students could drop from the alpha course and continue their progression with the next iteration.

During the alpha instance of the course, many issues were found with the course material. Most of the issues were typos and smaller mistakes in material, including ordering problems where students were required to complete exercises before the required tools or concepts were introduced. The significant problem was that some students did not have enough prerequisite knowledge or skills that were assumed.

The feedback led to the addition of reminder sections during the course on concepts that became relevant later, for example, ports (TCP/IP). As the alpha was only for a selected group of students, the changes included their suggestions, including new exercises and extending the material. In addition to the changes, the students had suggested the students were offered workshops twice in a week where students could ask instructors for help.

4.2.2 Beta instance

The course was released inside of the department of computer science to all students who wanted to participate. The prerequisites were on display but not enforced for students to join the course.

The second iteration had 81 students that completed at least 1 part. A large number of the students that attended workshops did give feedback, and after the beta course, it was evident that some of the subjects on the course were hard for students to understand. Some of the students got stuck in exercises, and some felt that the material did not offer enough guidance before the exercises. The biggest problem was the steep curve on the exercise difficulty. Some individual exercises could take hours because students had to learn a lot during that one exercise.

A large number of new exercises were created for the next instance to alleviate the long time spent on individual exercises. Some more experienced students asked for a new subsection in the material on multi-stage builds, and for this, a new exercise and a subsection was created. In total exercise counts changed from 9 in part 1 to 17, 6 in part 2 to 10 and 7 in part 3 to 8, almost doubling the number of exercises in part 1.

The attempt to fix the issues had changed the contents of the course in a significant way. The material and exercises were completed by a student who was not a part of the Department of Computer Science. This final review was done before the start of the open iteration of the course to avoid reintroducing grammar mistakes and to test the course contents in an open environment. The student was used as a representative of an average open university student with no experience of DevOps or command-line tools and minimal experience with programming. The skills were expected to reflect the skills of the open-source participant. The student completed the new exercises that were introduced between Beta and Open to find issues and iterate over the changes in course contents.

4.2.3 Open instance

The term open refers to the open university where this course was held. The course was open for everyone over the internet for free. Moreover, as the course was in English and not limited to Finnish students, it was advertised over social media. However, the course material had been in English during previous iterations, and as such, it does not impact the existing students at the University of Helsinki. The workshops that were held in the beta were not introduced in open iteration. They were cut to open more opportunities for students to follow and contribute to conversations on open forums such as the social media channel that was offered since the alpha.

For the open course, the difficulty curve of the course was significantly more gentle. The number of exercises was almost doubled for the first two parts. The intended effect was that students were less likely to get stuck on a challenging exercise as they had done a more accessible version of the similar exercise. As such, the overall difficulty of the course stayed the same.

4.3 Retrospective of the iterations

The choices made for the course format seemed to work. The course is entirely online, and the focus on exercises worked well with assignments being mostly unambiguous. Students were able to complete the course without external guidance and know if the assignment was completed based on its output.

Two notable changes were made to the course during iterations. The first change was when moving from alpha to beta. While it was intended for students to help their peers as the course started, there were only a limited number of peers for students to ask for help. To mitigate the absence of peer support at a weekly workshop was held in the second iteration. After the workshop, students had support continued to be available over the social media channels. This way, the instructor's role was slowly diminished from the 'sage on the stage' to 'the ghost in the wings' [49].

The second change was between beta and opened with increased exercise count to lower the difficulty curve of the course. In addition to this, as discussed previously, learning DevOps practices inherently requires facing different kinds of problems as students can not be expected to have much hands-on experience with them. The hypothesis on the addition of exercises is that the total time spent on this course

will not increase a lot since the overall difficulty stays the same.

The first two instances were held before the decision to start researching the subject, but the challenges existed regardless. The inclusion of exercises was the most significant difference between learning experiences between the original course and the university course. The feedback and iterations are documented as a part of the description of the course.

An important issue with the course is that many of the problems solved by DevOps may be somewhat abstract to students who have not worked on projects that require the skills that are essential for operations [39]. Predicting the requirements to get started with Docker is not trivial and as such students were not expected to have used or heard of containerization before joining the course.

5 Results

The following sections present the survey results. The survey offers a review of the attendees, their background, and their motivations for joining the course. Lastly, other university courses of the attendees who completed the course are compared to offer a more comprehensive approach to finding course prerequisites.

Of the 517 attendees who answered the survey at the beginning of the course none of the answers are removed from the results. The survey at the end of the course received 70 answers. There were a small number of attendees who left parts of the survey unanswered or had answers that could be regarded as outliers. Possible outlier answers were expected as the course was advertised through public social media channels. The total number of answers for each question are displayed as part of their respective tables as attendees may have skipped questions.

The latter survey included an open section with the following question: "Highlight three things you learned and three things that could be added to the course." The purposes of this last question were to map the areas that were important to attendees and to learn what may be missing from the material. Attendees were hoping for realistic service provider specific examples and guides on containerization. The feedback highlighted Google and Amazon container services as the platforms of interest.

An attendee did drop out in the latest iteration and gave feedback that they had difficulties with the course. The difficulties were not specified. Some of the feedback from attendees was directed at the chosen programming languages or methods in exercises. Possibly a more development-oriented approach to DevOps would alleviate the disconnect between the software and the attendee.

5.1 Background of course attendees

In Table 2 the questions are focused on the student background of the students. Of the attendees, about 50% were familiar with Docker or had used it before. As the course is an optional course, it is expected that the students that are interested in the course would join it. Although the rest had not used Docker many of them had a general idea of what containers are and how they are used.

Half of the attendees were students, and most of the completions were by students. The higher completion rate for students may be because the student attendees were

Table 2: Questions relating to student background with a survey at the beginning of the course and at the end of the course

Options	**Before %**	**After %**
I have used Docker before / Before starting this course I had used Docker		
Daily	4.41	3.13
A lot	4.01	4.69
Sometimes	15.43	21.88
A little bit	32.87	17.19
Never	43.29	53.13
Total #	**499**	**64**
My background is		
Student	42.25	51.52
Software engineer	17.10	13.64
Developer	16.50	18.18
DevOps engineer	6.24	4.55
Data scientist	4.23	0.00
Other	13.68	12.12
Total #	**497**	**66**
I have work experience of		
$>$ 6 years	24.75	27.27
3-6 years	13.48	10.61
1-3 years	23.54	25.76
$<$ 1 year	20.93	19.70
None	17.30	16.67
Total #	**497**	**66**
Docker or containers are used in my place of work		
Everything runs in containers	3.02	7.58
Almost everything runs in containers	11.90	12.12
Some	22.98	21.21
A little bit	12.70	6.06
Not used	17.74	16.67
I'm not employed	22.78	31.82
I cannot tell	8.87	4.55
Total #	**496**	**66**
I found this course via		
Work	6.41	
School	33.20	
Friend	9.51	
Social media	16.12	
Other	34.76	
Total #	**515**	

interested in credits or a certificate. As such, they did complete the course while the attendees that were not interested in course completion read or completed exercises that were of interest to them. The non-student attendees did not have an incentive for answering the last survey or completing enough exercises for course certificate or credits.

The background choice 'other' had a description field, and the answers ranged between multiple technical positions, the described positions in order of number of mentions were: systems administration, network engineer, data analyst, teacher or lecturer, and individual mentions from multiple other positions. Most of the attendees had at least some work experience.

Around 51% of the participants had a workplace in which containers were used at least "a little bit". This reinforces the claim that attendees joined the course with prior knowledge and personal interest in the subject. The number of attendees whose workplace is not using containers is low with around 18% of the attendees

Many of the students found the course via school. The students had a chance to fill in a description if they chose "other". A MOOC listing page of online courses with 13,3% of the total answers, as well as google search results with 7,8% of the total answers, were the most common other sources.

After the course the workplace of the attendee was using containers 4 percentages more likely. While at the same time the number of attendees with workplaces using containers only a little bit is significantly lower after the course. It may be that the workplace is adopting containerization into their workflows or that the attendee is personally enabling containerization there. But it's possible that attendees who had more experience with containers were more likely to complete the course.

5.2 Attendee motivation

In Table 3 the relationship between the course and the student is elaborated on. Students outlined their motivation and hopes for the course. The question regarding goals was a multiple-choice as the students had multiple motivations for joining the course. Personal improvement and work improvement were significant factors in joining the course. The course and academy results are lower as the attendees may not be students. Almost the same amount of attendees that are students were also interested in course credits.

As the number of students who answered the survey at the end of the course is

Table 3: Questions relating to student motivations with a survey at the beginning of the course and at the end of the course

Options	**Before %**	**After %**
My goals for this course (were)		
Learn about docker for work	74.85	63.77
Learn about docker for self	69.05	69.57
Learn about docker for academy	18.57	21.74
Course credits	35.98	53.62
Other	1.74	0.00
Total #	**517**	**69**
I'm looking forward to		
Learning how to use containers in production environment	68.35	
Learning how to run multi-container applications	58.45	
Learning how to containerize applications	65.24	
Overall learning about Docker	86.02	
Total #	**515**	

smaller than at the beginning, the completion rate for attendees interested in credits was higher. This relative increase in the interest in credits may be because student attendees who were interested in credits did complete the course to get them. While the attendees that were not interested in course credits only read/completed exercises that were of interest to them, hence not answering the last survey or completing enough exercises for course certificate or credits.

The four choices for what they were "looking forward to" were again a multiple choice where students could elaborate on their interests. The most critical shortcoming of the course is also seen with the expectation to learn how to use containers in a production environment as the course did not go as well in-depth on the use cases, methods or tools in production. The experience with external services in the course was quite limited as the only cloud service that was used was Heroku [50] even though the course had the potential to use other cloud service providers. However, most of the students were content with the overall Docker experience.

5.3 Attendee experience in containerization and DevOps

Table 4: Questions relating to student experience with a survey at the beginning of the course and at the end of the course.

Options	**Before %**	**After %**
I have a general understanding of containers		
I strongly agree	10.60	52.86
I somewhat agree	42.00	44.29
I neither agree or disagree	15.20	0.00
I somewhat disagree	17.20	1.43
I strongly disagree	15.00	1.43
Total #	**500**	**70**
I have understanding of the problems containers solve		
I strongly agree	11.20	55.71
I somewhat agree	48.40	41.43
I neither agree or disagree	17.60	0.00
I somewhat disagree	11.00	1.43
I strongly disagree	11.80	1.43
Total #	**500**	**70**
I'm familiar with DevOps or have general understanding what it is		
I strongly agree	11.49	20.00
I agree	45.56	61.54
I neither agree or disagree	21.98	15.38
I disagree	15.12	1.54
I strongly disagree	5.85	1.54
Total #	**496**	**65**

Some of the questions were designed to gauge the students' prior experience in the form of self-evaluation. The results would give an estimation of the skill level of the students who have joined the course. In Table 4, we see that a large number of students somewhat agreed to the notion that they had a general understanding of containers before the course started. It seems that the students who had prior knowledge of the subject were more likely to search and join the course. With the minimal advertising dedicated to the course, it was not entirely unexpected.

A general understanding of containers may mean that the person is familiar with the concept of containers, has created and managed containers. The latter question of what problems the containers solve indicates a limited knowledge of the area in the form of examples or read the material since the number of people agreeing to this statement is higher than in the former. It is in line with the previous suggestion that students joined the course if they knew something about the subject before.

As the definition of DevOps is not trivial, it was of interest to ask the students what they thought about their familiarity with the concept. A more in-depth review of the results revealed that attendees with more work experience were more likely to agreed that they were familiar with DevOps or had a general understanding of what it is. For all of the questions, a possible influence on the higher rate of agreement may also stem from the Dunning-Kruger effect with students overestimating their expertise [51]. An attendee answered with strongly disagreeing on every point. This attendee is expected to be the same person who answered exceptionally high numbers to the time spent in each part.

5.4 The rate of completion and time spent

Most of the constructive feedback was given in relation to exercises. Difficulties in exercises may indicate shortcomings in the course and not the exercise. This is backed up by the deviation between attendees with more or less experience; the exercises were extremely challenging for beginners. To solve this, the number of exercises in the material was higher in later iterations. The additional exercises work as a stepping stone for the more challenging ones. Because of this, the average amount of time spent on exercises did change from the course instances.

Table 5: Student part completion rate as a part of the survey

Options	**After**
I completed the following parts	
Part 1	66
Part 2	51
Part 3	45
Total #	**67**

In Table 5, we see that the attendees mostly completed every part in the course even

though attendees had the option to cut the course completion short. Attendees could use the offered course to fill in their final credits in case they needed one or two course credits. This is because the course had variable amount of credits available based on the amount of parts completed and most of the other courses offered in local universities were of 5 credits. It was a known choice for students to round their credits with the least amount of work.

Table 6: Student time spent (in hours) on each of the course parts with average, median and mode of time spent in each part and the number of submissions

Iteration	**Alpha & Beta**	**Open**
Part 1		
Average	9.8	12.4
Median	8.0	8.0
Mode	8.0	10.0
Submissions	81	75
Part 2		
Average	8.0	12.8
Median	7.0	8.0
Mode	5.0	6.0
Submissions	64	57
Part 3		
Average	8.8	12.0
Median	7.0	10.0
Mode	10.0	10.0
Submissions	32	37

Table 6 shows the amount of time spent in each part by students and the number of submissions in each part. The table includes attendees that self-reported exceptionally high numbers of hours spent in the exercises, over 30 for each part. Excluding these outliers, as the number of exercises was increased the time spent on exercises did not increase dramatically. Students were no longer getting stuck on exercises that were requiring more experience than they had. Instead, they faced exercises that were on their level.

5.5 Identifying course prerequisites

To learn more about the student population participating in the course an inspection into the past studies of students that completed the course was conducted. Large differences in the completion rate of other courses could reveal prerequisites or indicate other connections between course completions.

Table 7: Students with course completion and the most common courses by students that completed part 3

Part	**Students #**	**Students %**
Full Stack Web Development		
Part 1	18	72
Part 2	32	82
Part 3	62	83
Introduction to Data Communication		
Part 1	18	72
Part 2	27	69
Part 3	41	55

Table 7 does not include students who dropped out, only completions. For all of the courses naturally, 100% of students had completed DevOps with Docker. Totaling 25 students with 1 credit, 39 students with 2 credit completion and 75 students with 3 credits. This includes all students who have received credits from the course.

Most completion percentages between students were equal or had minimal differences as most of the students had completed first-year studies in computer science. The two courses chosen for the Table 7 were of most interest. Full Stack Web Development is an optional course but an exceptionally large number of attendees had completed it. Whilst Introduction to Data Communication was initially suspected to be a prerequisite for this course. The negative correlation between the completion of the DevOps with Docker and Introduction to Data Communication is not discussed further.

Most of the students had completed at least the first year of computer science studies including introduction to programming and even second year studies such as web software development courses. These courses contain the expected prerequisites for the DevOps with Docker course.

6 Discussion

The discussion will start by outlining the research questions and the methods used to answer them. After which the research questions are answered with the results. In addition, an overall view of the design and state of the new course is presented. The threats to validity are presented in the final subsection.

6.1 The current state of DevOps education

The first research question "How are skills related to DevOps taught now?" is answered via a review of the software engineering curriculum of the department of computer science at the University of Helsinki. Target for the review is the contents of the courses learning goals.

Analysis of the courses learning goals alone does not present a reliable source of information as the state of the written goals varies and does not always reflect on the contents of the course. To support the estimation for the state, the previous research done focusing on the University of Helsinki [4] and personal experience from the Master's studies of the software engineering track was taken into consideration. With the three sources of information, it can be argued that the level of operations education on the university level is lacking at the University of Helsinki.

The core concepts that operations deal with are mentioned, for example, during an introduction to the software lifecycle phases, but are neither confronted or taught in-depth. Software engineering lab is the exception to this as teams of students are interacting in a semi-realistic professional environment working through most of the phases of the software lifecycle. Outside the single course, as DevOps is bridging the gap between development and operations with various methods, the situations in which the practices can be utilized are rare.

Analysis based on the courses learning goals at the University of Helsinki is not sufficient to draw conclusions of the overall state of university education, but it presents the context in which the following research questions are answered in.

6.2 Prerequisites for learning Docker

For the second research question, "What are the prerequisites to start learning Docker in university?" a course teaching Docker was analyzed. The exercises in

Docker would require knowledge of the Linux operating system and application networking, which are not explicitly taught to students on mandatory courses. Except for Linux and its general usage are introduced during a 1 ECTS course, Computing tools for CS studies, in the first period of computer science studies. No skills were set as prerequisites for joining the course, but a general understanding of software development and experience with a command-line interface of choice were expected from attendees. This was an estimation of the bare minimum requirements to get started with the course. The initial expectation was that the course would require much time from students.

During the iterations of the course, it was evident that the prerequisites were not as high as anticipated. Students with minimal experience in both development and operations were able to complete the course. The first two iterations were held for students in the Department of Computer Science and had a lower average time spent on part. As the course was marketed with minimal prerequisites through the open university, it gathered a more varying level of attendees. The average time spent per part by students in the final iteration had a larger deviation, but not significantly so. Overall, the feedback has been positive during the final iteration.

Of courses completed by students previously, it is notable that the students who completed more parts were more likely also to have completed courses on web development. This may be because most of the examples contained web software or because the subject, containers, shows many of its benefits in web software. The difference in completion rates indicates that the familiarity of the working context, web software, may improve student motivation.

Another possible reason for the higher completion rate for the attendees on web development is because the courses are MOOCs and use a similar method of self-assessment for course work. The students were more familiar with the time and work requirements as well as the working practices of online courses. In the end, the minimal prerequisites for university students to learn DevOps practices were found to be nonexistent.

6.3 Exercises aiding in learning DevOps practices

The third research question "What kinds of exercises can be offered to students to enforce learning DevOps practices?" was also investigated using the survey for students attending the course. The chosen exercise method had students learning

key skills via small exercises that supported students expanding a single application that grew in complexity as students added functionality to it. The functionality was not done by coding but by adding various parts into the web application by using separate containers in a network. From students' feedback, this method was considered to be suitable.

During the first two iterations of the course, before offering the students through open university, more exercises were added. On the workshop sessions offered during the second iteration of the course, feedback was that some of the exercises were difficult. Second iteration was inviting a broader range of students of different experience and skill levels within the department of computer science. To mitigate this skill difference, the course leaned towards more instructional scaffolding with an increased number of standalone exercises that had little dependencies on other exercises but prepared students for the implementation of the techniques on the main exercise application. As the mode of time spent per part hardly increased while average and median did even while adding a large number of exercises. The increase in exercise volume may have the opposite effect on the time spent on each exercise.

Another feedback that was prominent in the after-course survey and in the motivations was the desire for production-like environments, namely Kubernetes. It could be possible to add more applications or to increase the complexity of the application so that the exercises are in a more realistic context. DevOps was taught to students as a way to get familiar with the concepts used in the industry, but the feedback suggests that problem based learning in a cloud environment could be a potential method as well.

6.4 The core challenges for students

The final research question "What are the core challenges for students when learning DevOps practices?" is closely related to the previous question on the exercises offered to students. The core challenges are based on the written feedback in the surveys and discussion over the social media channel.

Most prominently, students mentioned challenges that were related to areas such as the networking of containers, and the rest were from the tools or operating system used with the examples and exercises on the course.

Networking was the most significant challenge in the alpha and beta instances for the

students. In contrast to the mandatory networking course where students learn e.g. how the internet works and what are the layers in the OSI model, the DevOps course focused only on the transport layer with port configuration between applications and containers.

The usage of operating systems, namely Linux, during the course was challenging as many students were not familiar with the usage of the command-line interface and, for example, creating small script files using bash. The usage of the command-line was mentioned as part of the prerequisites for the course.

6.5 Reflecting course design

The goal and expectation was that students could complete the course without significant difficulties. The learning goals were outlined to students at the beginning of the course:

- 1 part: The student has learned what Docker is and knows how to use it in different scenarios.
- 2 parts: The student has a general understanding of docker-compose and is able to implement advanced Docker configurations.
- 3 parts: The student has a deeper understanding of the Docker ecosystem and its use cases.

The anticipated challenges matched reasonably well with the challenges that were met during the first iterations of the course. Students required a lot less time than expected, with experienced students completing parts in less than 2 hours.

Communication between containers was a continuous challenge for students in alpha and beta iterations of the course. The amount of feedback regarding difficulties in networking decreased after adjusting the course to have a more significant emphasis on exercises.

Another problem that exists with the course is that however close it is to both development and operations, it was not able to encompass the whole extent of DevOps. Students did not develop their own software during the course. For future iterations, it would make sense to build at least a small program with students that they could deploy with the DevOps practices. However, this may introduce new challenges that have not been tackled yet. Neither development or operations were

completely explored for the purposes of limiting the course subject area to a suitable size.

Overall, students' feedback in the final iteration of the course was positive, with multiple students crediting the course for summer jobs or trainee positions that they had achieved, or mentioned that they had started to use the learned techniques. As the course was initially designed for students to get into the software development industry, this was seen as a great success. However, the remaining problems are still significant; a high number of students that completed the course requested there to be more in-depth exercises about the tools used in production environments such as Kubernetes [52], a container orchestration platform that was initially released in 2014 and has since gained popularity.

6.6 Validity

This section is split into two subsections. Inspections into the internal validity and external validity are presented under these subsections by following the list of threats to validity presented by W. Shadish et al. [53]. The following paragraphs before the first subsection present limitations and validity of the overall context. The context in which the course was designed and research was done.

Personal experience with teaching students is limited, with no higher-level education on teaching. The choice to design the course in the way it discussed in previous chapters was not made with prior formal teaching education. This lack of experience imposes the possibility of not seeing other solutions that could solve the same problems and possibly choosing worse solutions than what we could have.

The course deadline was several months after the submission of this thesis. Students had no incentive to complete the open version of the course by the time this thesis was submitted. The number of students who completed the open instance of the course is limited to students who completed the course in the first eight months of the 9-month course.

6.6.1 Internal validity

The internal validity is assessed via the threats to internal validity: ambiguous temporal precedence, selection, history, maturation, regression, attrition, instrumentation, and additive and interactive effects of threats to internal validity [53].

Ambiguous temporal precedence: students were able to join, progress, and complete the course at their own pace without time limitations. The two surveys were offered at the beginning and end of the course to avoid mistaking progress on prior knowledge.

Selection: students were of two different sets that overlap. The open course was available for free to anyone who was interested and found the course. The previous iterations were available only for the students at the department of computer science at the University of Helsinki. Without limitations of study periods or time; otherwise, the total selection of students was a set of anyone interested in the course subject. A significant portion of course attendees had prior knowledge of containers and as such, were not having problems with the exercises. Without considering this, it would affect the level of prerequisites and the relative number of challenges that students had.

History: students were able to join, progress, and complete the course at their own pace without time limitations. There is potential for cases in which students first joined the course, answering the beginning survey after which doing three months of professional work with the technologies — then finishing the course answering the final survey.

Maturation: students not having time limitations led to maturation between the surveys. A student may progress on other courses as they are attending the course that was inspected. The maturation of students does not have a significant effect on the research questions. The students are inspected as a population where required prerequisites and core challenges emerge. More advanced students may not give feedback on the challenges as they had none.

Regression: the selection method and conclusions drawn for research questions are seemingly not susceptible to possibility regression to mean, but without a control group to compare the attendees to, there are no conclusive means of verifying this. The issue of missing a control group is also a general problem that may be added to almost any threat to validity.

Attrition: the attendee drop between surveys is presented as part of the results. There are various reasons for students to succeed, for example, prior experience with the online platform and methods of working in online courses or motivation that was gained from employer encouragement. The attrition rate does not affect the validity of conclusions related to research questions.

Testing: this threat is not considered as the course is intended to be completed once.

Instrumentation: the course iteration changed instruments. The course continuously changed between iterations, and surveys were only offered to students of the final iteration. The problem with the methods used to gather data is that participants could only be estimated across the two surveys. Precisely following the answers between surveys was not possible because the student session was not preserved, and no identifier was given to students to make sure the questions were answered both during the start and when ending the course. The result was that while there was a large number of answers for the first survey, there was no information on which attendees also answered the latter survey.

Additive and interactive effects of threats to internal validity: the total bias is challenging to estimate. There are also other threats to internal validity that result from mistakes in research methods.

A mistake in the survey design was that some of the questions are not unambiguous: "work experience" can be interpreted to refer to any work experience rather than software development industry experience. However, this did not prevent estimating the starting levels of students and their backgrounds. This was marginally mitigated with personal feedback from attendees.

6.6.2 External validity

The external validity is assessed via the threats to external validity: interaction of the causal relationship with units, interaction of the causal relationship over treatment variations, interaction of the causal relationship with outcomes, and context-dependent mediation [53].

Interaction of the causal relationship with units: the presented course focuses on a tool used in DevOps. The benefits of exercise focused courses may not be similar in other courses. Even in a course that focuses on DevOps tools, more complex tools may require more time spent and more resources on material, as opposed to the chosen method of focusing on exercises and examples. As an example, another DevOps tool such as Kubernetes or Jenkins [54] can offer entirely new challenges even in a similar format.

Interaction of the causal relationship over treatment variations: the course was offered as an optional course. The generalization of the results to a mandatory course of university-level may present difficulties as the data offered here contains a

large portion of students that had prior experience of the course subject. Even as course credits were a significant motivation, on a mandatory course, the number of unmotivated students may be higher, and the results vary.

Interaction of the causal relationship with outcomes: an example outcome may be that attendees have a higher level of skill compared to students that did not complete the course. However, with the lack of a control group, this remains an open question.

Interactions of the causal relationship with settings: the setting for generalization is limited to the university level. As the chosen method of online course is relatively limited, generalization to a lecture course is open to consideration. The chosen format for the course prevented group work. The goal of DevOps bringing the development and operations teams closer to each other with collaboration is lost in the course format as it does not include collaboration with others. The course, as such, does not offer a complete picture of DevOps. As such, the solutions offered for DevOps can not be generalized.

Context-dependent mediation: the context dependency for the results is the university level and as such limited to university level attendees and university level courses.

6.7 Future work

The course was not able to encompass the whole extent of DevOps; students did not develop personal software during the course. For future iterations, it would make sense to build at least a small program with students that they could deploy with the DevOps practices. However, this may introduce new challenges that have not been tackled yet. The focus in the course was in the software lifecycle stages right at the edges of development and operations to limit the course subject area to a suitable size.

Teaching DevOps practices does not contain in any way new methods of teaching, but practical approaches seem to be a significant choice to teaching it. It is necessary to gather more reference material from courses on the same or similar topics to verify the effectiveness. If a more theoretical approach seems worthwhile and is arranged, it is possible to gain more insight into the validity of practical approaches that were used here.

For similar courses focused on the practicalities, it would be essential to gather more data from students after the completion of the course. This requires investigation

if students are gaining significant benefits from the course and are the skills applied to other areas of software development or research.

For the University of Helsinki, there is still a need to improve the curricula. The Department of Computer science contains software engineering program which should contain every aspect of software engineering from planning to maintenance [3]. As such, it should be considered that the University of Helsinki expands software level courses from development to operations as well, even if it would lead to more practical courses compared to purely academic courses.

Students have a limited amount of time to invest in the course, and as such, the course did not include delivery to production with a delivery pipeline. Expanding the contents of the course to include other DevOps tools, such as Kubernetes, was requested by students. The curriculum can be expanded with a course delving into the depths of production environments and workflows, focusing on the deployment and monitoring of software to answer the requests.

7 Conclusions

Deployment, operation, and maintenance are part of software engineering, and the education regarding them lacks at the university level. DevOps education is offered to solve the problem as it is joining both development and operations as a cross-functional methodology. Teaching DevOps is challenging, and the problems revolve around the nature of DevOps. The requirements in both teaching and learning include operations experience, and some of the benefits that following DevOps methodologies offers may not be apparent in a limited environment, such as in a course context. An exception to this may be courses in which a group of students collaborate on a larger project with students having the responsibility for deployment and maintenance during the course.

The four research questions emerged from the challenges of implementing a course introducing containerization. The first of which, "How are skills related to DevOps taught now?", was answered by reviewing the current state of DevOps education in the department of computer science at the University of Helsinki. All of the courses included in the software engineering track were compared to the definition of DevOps using the courses' learning goals to analyze the current state. The conclusion was that as the master's program in software engineering is relatively limited in the number of courses that concern the operations side of the software lifecycle, there is no emphasis on DevOps practices even if they were mentioned.

For the following three questions, "What are the prerequisites to start learning containerization as a DevOps practice in university?", "What kinds of exercises can be offered students to enforce learning DevOps practices?" and "What are the core challenges for students when learning DevOps practices?", a course was created and iterated upon to answer them. The course was named "DevOps with Docker" as the tools chosen for containerization are Docker and docker-compose. During the course iterations, student feedback was collected via two surveys. Some of the feedback that influenced the course design during iteration was given via personal feedback and discussion with students.

The prerequisites to start learning containerization as a DevOps practice in university were investigated using the previous courses students had completed before joining the Docker course. Students had hardly any problems with the course subject after adjusting the course to have a more considerable emphasis on exercises. The conclusion was that there is nothing special about the learning subject of Docker

for students, in contrast to the expectation that students would have difficulties in adopting a new technology that's crossing over from development to operations. The online format had its own challenges and limitations. Students with more online courses and web software experience were likely to complete the full course.

The offered course was primarily exercise focused containing instructional scaffolding. Students completed the exercises with described completion requirements on their personal computers. The primary goal was for students to use Docker and docker-compose to build an increasingly complex application consisting of multiple smaller applications serving different purposes. DevOps toolchains offer an entryway into the DevOps mentality and exercises that focus on the tools, and their usage is a viable method for learning the practices used in DevOps. The students' perspective was that there was no need for additional instructional material to complement the exercises.

The core challenges the students faced were also evaluated with the survey, and the personal feedback received from students. Students voiced difficulties with command line usage and networking with containers during the first two iterations of the course. In the final iteration, the students were requesting more realistic, production-like environments and toolsets to practice. The online format and web software focus can also be considered to be challenging as the students with more experience in web software development completed more parts of the course. The extended skill range of both development and operations requires an introduction to more practical approaches to otherwise familiar concepts such as networking and operating systems.

References

1 A. Wiedemann, M. Wiesche, and H. Krcmar, "Integrating Development and Operations in Cross-Functional Teams - Toward a DevOps Competency Model," in *Proceedings of the 2019 on Computers and People Research Conference - SIGMIS-CPR '19*, (Nashville, TN, USA), pp. 14–19, ACM Press, 2019.

2 R. Jabbari, N. bin Ali, K. Petersen, and B. Tanveer, "What is DevOps?: A Systematic Mapping Study on Definitions and Practices," in *Proceedings of the Scientific Workshop Proceedings of XP2016*, XP '16 Workshops, (New York, NY, USA), pp. 12:1–12:11, ACM, 2016.

3 ACM Computing Curricula Task Force, ed., *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM, Inc, Jan. 2013.

4 M. Luukkainen, A. Vihavainen, and T. Vikberg, "Three years of design-based research to reform a software engineering curriculum," in *Proceedings of the 13th Annual Conference on Information Technology Education*, SIGITE '12, (New York, NY, USA), pp. 209–214, ACM, 2012.

5 H. B. Christensen, "Teaching devops and cloud computing using a cognitive apprenticeship and story-telling approach," in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pp. 174–179, ACM, 2016.

6 "Instructional Scaffolding in Online Learning Environment: A Meta-analysis - IEEE Conference Publication." https://ieeexplore.ieee.org/abstract/document/6821832.

7 L. Chen, "Continuous Delivery: Huge Benefits, but Challenges Too," *IEEE Software*, vol. 32, pp. 50–54, Mar. 2015.

8 T. Mikkonen. Personal Communication, 2019.

9 M. Rajkumar, A. K. Pole, V. S. Adige, and P. Mahanta, "DevOps culture and its impact on cloud delivery and software development," in *2016 International Conference on Advances in Computing, Communication, Automation (ICACCA) (Spring)*, pp. 1–6, Apr. 2016.

10 S. Dart, “Concepts in Configuration Management Systems,” in *Proceedings of the 3rd International Workshop on Software Configuration Management*, SCM ’91, (New York, NY, USA), pp. 1–18, ACM, 1991.

11 “Git.” https://git-scm.com/, 2019. Accessed: 2019-10-29.

12 “Ansible.” https://www.ansible.com, 2019. Accessed: 2019-10-29.

13 “Puppet.” https://puppet.com/, 2019. Accessed: 2019-11-10.

14 D. Spinellis, “Don’t Install Software by Hand,” *IEEE Software*, vol. 29, pp. 86–87, July 2012.

15 “What is a Container?.” https://www.docker.com/resources/what-container, 2019. Accessed: 2019-10-29.

16 K. Hightower, B. Burns, and J. Beda, *Kubernetes: Up and Running: Dive Into the Future of Infrastructure*. "O’Reilly Media, Inc.", Sept. 2017.

17 J. Horowitz, “Configuration Management is an Antipattern.” https://hackernoon.com/configuration-management-is-an-antipattern-e677e34be64c.

18 A. M. Joy, “Performance comparison between Linux containers and virtual machines,” in *2015 International Conference on Advances in Computer Engineering and Applications*, pp. 342–346, Mar. 2015.

19 W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated performance comparison of virtual machines and Linux containers,” in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 171–172, Mar. 2015.

20 M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. D. Rose, “Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments,” in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pp. 233–240, Feb. 2013.

21 M. H. Syed and E. B. Fernandez, “A reference architecture for the container ecosystem,” in *Proceedings of the 13th International Conference on Availability, Reliability and Security - ARES 2018*, (Hamburg, Germany), pp. 1–6, ACM Press, 2018.

22 H. Kang, M. Le, and S. Tao, "Container and Microservice Driven Design for Cloud Infrastructure DevOps," in *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 202–211, Apr. 2016.

23 C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "Devops," *IEEE Software*, vol. 33, pp. 94–100, May 2016.

24 F. Brooks, Jr, "No silver bullet essence and accidents of software engineering," *IEEE Computer*, vol. 20, pp. 10–19, 04 1987.

25 A. Wiedemann, "A New Form of Collaboration in IT Teams - Exploring the DevOps Phenomenon," *PACIS 2017 Proceedings. 82.*, 2017.

26 Y. Zhang, B. Vasilescu, H. Wang, and V. Filkov, "One size does not fit all: An empirical study of containerized continuous deployment workflows," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*, (Lake Buena Vista, FL, USA), pp. 295–306, ACM Press, 2018.

27 C. Boettiger, "An Introduction to Docker for Reproducible Research," *SIGOPS Oper. Syst. Rev.*, vol. 49, pp. 71–79, Jan. 2015.

28 R. D. Peng, "Reproducible Research in Computational Science," *Science*, vol. 334, pp. 1226–1227, Dec. 2011.

29 "Docker." https://www.docker.com/, 2019. Accessed: 2019-10-29.

30 D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, pp. 81–84, Sep. 2014.

31 J. Pengxiang and P. Leong, "Teaching Work-Ready Cloud Computing using the DevOps approach," in *Proceedings of International Symposium on Advances in Technology Education*, Sept. 2014.

32 K. Kuusinen and S. Albertsen, "Industry-Academy Collaboration in Teaching DevOps and Continuous Delivery to Software Engineering Students: Towards Improved Industrial Relevance in Higher Education," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pp. 23–27, May 2019.

33 L. Haaranen and T. Lehtinen, "Teaching git on the side: Version control system as a course platform," in *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '15, (New York, NY, USA), pp. 87–92, ACM, 2015.

34 "Circleci." https://circleci.com/, 2019. Accessed: 2019-10-29.

35 A. Radermacher and G. Walia, "Gaps between industry expectations and the abilities of graduates," in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education - SIGCSE '13*, (Denver, Colorado, USA), p. 525, ACM Press, 2013.

36 A. Radermacher, G. Walia, and D. Knudson, "Investigating the skill gap between graduating students and industry expectations," in *Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014*, (Hyderabad, India), pp. 291–300, ACM Press, 2014.

37 "IEEE Standard for Configuration Management in Systems and Software Engineering," *IEEE Std 828-2012 (Revision of IEEE Std 828-2005)*, pp. 1–71, Mar. 2012.

38 N. Kerzazi and B. Adams, "Who Needs Release and Devops Engineers, and Why?," in *Proceedings of the International Workshop on Continuous Software Evolution and Delivery*, CSED '16, (New York, NY, USA), pp. 77–83, ACM, 2016.

39 C. Jones, "A Proposal for Integrating DevOps into Software Engineering Curricula," in *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment* (J.-M. Bruel, M. Mazzara, and B. Meyer, eds.), Lecture Notes in Computer Science, pp. 33–47, Springer International Publishing, 2019.

40 A. Begel and B. Simon, "Novice software developers, all over again," in *Proceeding of the Fourth International Workshop on Computing Education Research - ICER '08*, (Sydney, Australia), pp. 3–14, ACM Press, 2008.

41 C. Anderson, "Docker [software engineering]," *IEEE Software*, vol. 32, pp. 102–c3, May 2015.

42 M. Paksula, "docker.md." https://gist.github.com/matti/0b44eb865d70d98ffe0351fd8e6fa35d, 2017. Accessed: 2019-11-27.

43 D. Glance, M. Forsey, and M. Riley, "The pedagogical foundations of massive open online courses," *First Monday*, vol. 18, no. 5, 2013.

44 D. Clark, "MOOCs: Taxonomy of 8 types of MOOC," 2013. Accessed: 2019-10-29.

45 A. Sachdeva, P. K. Singh, and A. Sharma, "MOOCs: A comprehensive study to highlight its strengths and weaknesses," in *2015 IEEE 3rd International Conference on MOOCs, Innovation and Technology in Education (MITE)*, pp. 365–370, Oct. 2015.

46 "Overview of Docker Compose." https://docs.docker.com/compose/, Aug. 2019.

47 J. Singer, T. Lethbridge, N. Vinson, and N. Anquetil, "An Examination of Software Engineering Work Practices," in *CASCON First Decade High Impact Papers*, CASCON '10, (Riverton, NJ, USA), pp. 174–188, IBM Corp., 2010.

48 M. Galesic and M. Bosnjak, "Effects of Questionnaire Length on Participation and Indicators of Response Quality in a Web Survey," *Public Opinion Quarterly*, vol. 73, pp. 349–360, Jan. 2009.

49 M. Mazzolini and S. Maddison, "When to jump in: The role of the instructor in online discussion forums," *Computers & Education*, vol. 49, no. 2, pp. 193–213, 2007.

50 "Heroku." https://heroku.com/, 2019. Accessed: 2019-11-17.

51 D. Dunning, "Chapter five - the dunning–kruger effect: On being ignorant of one's own ignorance," vol. 44 of *Advances in Experimental Social Psychology*, pp. 247 – 296, Academic Press, 2011.

52 "Kubernetes." https://kubernetes.io/, 2019. Accessed: 2019-10-29.

53 W. R. Shadish, T. D. Cook, and D. T. Campbell, *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*. Wadsforth Cengage Learning, 2002.

54 "Jenkins." https://jenkins.io/index.html, 2019. Accessed: 2019-11-28.