Teaching DevOps: A Tale of Two Universities

Richard Hobeck
Ingo Weber
Software and Business Engineering
Technical University of Berlin
Berlin, Germany
firstname.lastname@tu-berlin.de

Len Bass
Hasan Yasar
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
lenbass@cmu.edu
hyasar@cmu.edu

Abstract

DevOps is a set of practices in software engineering that is in high demand by industry. It is a dynamic field which constantly adds new methods and tools. Teaching DevOps prepares today's computer science students for best-practices in a working environment but challenges university lecturers to provide central concepts while staying up-to-date with current trends. In this paper we reflect on our experiences teaching DevOps at two universities (in the USA and Germany) in an inverted classroom format. We describe how we set-up the courses, provide a brief analysis of data we collected, and share our lessons learned.

CCS Concepts: • **Software and its engineering → Agile software development**.

Keywords: DevOps, inverted classroom, software engineering

ACM Reference Format:
Richard Hobeck, Ingo Weber, Len Bass, and Hasan Yasar. 2021. Teaching DevOps: A Tale of Two Universities. In *Proceedings of SPLASH-E '21, Chicago, IL (SPLASH-E '21).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

DevOps has become a widely established set of practices in software development. A central goal of DevOps is to accelerate the time-to-market of software features [2]. Including DevOps as a subject in IT-related university studies can provide students with the basics to work on agile software engineering projects with fast release cycles, as well as promote a broader adoption of DevOps in practice. A teaching method that benefits from the wide availability of digital technology in education is the inverted classroom: students familiarize themselves with the learning content outside the lecture (e.g., by means of videos) and reproduce and apply their knowledge to problems posed in synchronous classroom lectures [3]. The authors of this paper synchronized the content of their DevOps classes and taught them in an inverted classroom format. The classes were given at two different universities: Carnegie Mellon University (CMU) in Pittsburgh, USA and Technische Universität Berlin (TUB) in Germany. In this paper we describe and compare the teaching practices of the two classes at CMU and TUB, and share our experiences with inverted classroom as a pedagogic practice in software engineering subjects.

In the model of learning by Hughes, Toohey, & Hatherley [4], classic lectures and, in our context of inverted classroom, video lectures are useful for the first two phases (mostly 1), tutorials for phases 2-4 (mostly 2), and assignments and midterm exams for phases 3 and 4. Phase 5: reflect and adjust, is addressed through the in-depth discussions with the class; in more traditional teaching styles, this phase is often not addressed.

In the following, we will first give an overview on the course contents, structures, and teaching methods. Subsequently, a brief analysis of time sheets will present how much time students at the two universities spent on the assignments in Section 3. Then we report on our lessons learned and other experiences in Section 4. Finally, we discuss threats to validity and conclude in Section 5.

2 Teaching modalities

DevOps comprises a wide field in computing and information systems, integrating *organizational*, *cultural* as well as *technical* aspects. We teach at engineering departments at two universities and thus focused largely on the technical facets of DevOps in our courses. That is also reflected in the targeted learning outcomes of the course. On the one hand, students will be familiar with technical concepts such as microservice architecture, deployment pipelines or infrastructure as code (see the course content for CMU and TUB in Table 1). On the other hand, we offer practical assignments with industry-standard software tools such as *Docker*[1],

[1] https://www.docker.com/

SPLASH-E '21, October 17–22, 2021, Chicago, IL

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Course Content	CMU	TUB
Introduction to DevOps	R	R
Infrastructure as code	R	O
Configuration management	R	O
Virtual machines	R	R
Containers	R	R
Networking	R	O
The cloud	R	O
Container management	R	R
Infrastructure security	R	R
Deployment pipeline	R	R
Microservice architecture	R	R
Service meshes	R	R
Postproduction	R	R
Disaster recovery	R	R
Secure development	R	O
Domain specific DevOps	R	O
Distributed system architecture	R	O
Case studies	R	R

Table 1. Comparison of course contents at CMU and TUB (R: required content, O: optional content)

Kubernetes[2], *Jenkins*[3], or *Logstash*[4]. The content taught is aligned with the learning outcomes, and so are the forms of assessment, including formative (quizzes, assignments) and summative (exams / tests).

However, the two courses are offered at different universities and are taught in conformance to the specifics of the educational system they are embedded in, which will be covered in the next two subsections.

2.1 CMU Course

The CMU DevOps course is divided into two portions – equal in assessment weight. These two portions are theory and practice. The theory is taught using a form of "inverted classroom". The practice is taught through assignments using various tools important in the DevOps world. The theory is divided into 25-30 video lectures that are 20-30 minutes long. The lecture videos as well as the lecture slides are publicly available on the platforms PresentationTube[5], YouTube[6], Slideshare[7], and GitHub[8]. The number of video lectures offered per semester at CMU depends on holidays and the university schedule. Prior to class a student will:

- Watch the video for that day
- Read the assigned reading – mostly from the textbooks [1, 2], but also from other sources.
- Post one question for in class discussion

During class, 15 minutes are allocated for an automatically graded quiz over the previous class's video content, discussions, and readings. This is followed by a discussion of the questions in the quiz, usually lasting another 10-15 minutes. There will then be a discussion based on the video, readings, and posted questions. The length of the discussion is based on class size. It may take up the remaining time. If it does not, there is a breakout session with 5-6 students per breakout group over some topic. It may involve answering some questions or it may involve developing a context diagram for some piece of software relevant to an assignment or the lecture topics.

Outside of class the students will perform the assignments, individually. There are seven to ten assignments per semester. Each assignment involves installing a tool and using that tool to perform some simple task. Students are asked to hand in step-by-step instructions that explain their approach, screenshots, and a textual response to a question posed over the assignment. We further ask students to provide simple time sheets, where they recorded how much time they spent on the assignments. Students use the internet to educate themselves about the use of the tool. Questions about the assignment are asked and answered via e-mail or in laboratory sessions.

2.2 TUB Course

At TU Berlin, the course content was largely designed as a subset of the one taught at CMU. However, in contrast to CMU, which included two time slots of 1.5 hours per week, the TUB course is block-structured: each block is 3.5-4 hours in length (including breaks), and takes place every two weeks. In the first block, the lecturer presents an overview of the core concepts of DevOps. Subsequent blocks start with a quiz of 10-20 minutes each, followed by discussions about the video contents. Mostly this is done with discussion questions, which – in contrast to CMU – were prepared by the lecturer ahead of time (and not by the students). During the discussion, notes are taken by the lecturer, while the screen is shared with the class. Often points raised by students are refined through follow-up questions and corresponding answers. To this end, we use a note-taking tablet, on which the lecturer writes and draws by hand. PDFs of the resulting notes are later shared with the class. In one to two blocks, guest lectures are given by industry professionals.

At TU Berlin, students do not select their courses before the semester, but at the start of it. In TUB's moodle installation[9], interested students self-enroll for courses that meet their interest; for DevOps, the first offering registered over 220 such enrollments, though only 25 spots are offered (to

[2]https://kubernetes.io/
[3]https://www.jenkins.io/
[4]https://www.elastic.co/de/logstash
[5]https://presentationtube.com/users/profile/publicprofile.php?m=lbass
[6]https://www.youtube.com/user/lenjbass
[7]https://www.slideshare.net/lenbass/presentations
[8]https://github.com/cmudevops/course-materials
[9]https://isis.tu-berlin.de/

enable the kind of active participation we aim for). Thus, we implemented the following selection procedure. First, a kick-off meeting was scheduled where the lecturer explained the course structure, content (a 15-minute presentation), grading components, and expectations. Students then could opt to apply for the 25 spots, and the chair offering the course selects 25 students on the basis of formal criteria and randomness, as per the rules of the university. This student selection introduces a difference to CMU's set of participating students, since TUB students have more information but also need to make an effort to become a participant of the course.

The course at TUB also includes a hands-on part that consists of four to five assignments. The assignments are a subset of the assignments offered at CMU (except Terraform) with the same instructions and deliverables. They are due every two weeks and are accompanied by Q&A-sessions and short overview tutorials for the students, but the largest portion of necessary knowledge to complete the assignments has to be self-taught. Additionally, we responded to student's questions in moodle and via e-mail.

3 A brief analysis of time sheet data

The assignments offered to the students of both universities, CMU and TUB, were drawn from the same pool of assignments. The instructions were kept the same in every semester (Jenkins remained fixed after summer 2019) slight changes were made to supportive material, as the tools changed over time. The students' task in each assignment comprised solving small software challenges with well-established DevOps tools. For all assignments, the students were asked to document the time it took them to complete the tasks. The assignments were related to the following software tools or techniques: virtual machine management, SSH, Chef, EC2, Chef Vault, Vagrant, Jenkins, Docker, Nagios, Kubernetes, Ansible, Ansible Vault, Wire Shark, Terraform, and Logstash. The times were documented in the academic terms from fall 2017 to summer 2021 at CMU and summer 2020 to summer 2021 at TUB.

Figure 1 shows a comparison between the median times recorded from CMU and TUB students, grouped by assignment once they were stable. Note that not all assignments were offered in every semester or each university. The figure shows that assignments that were offered at an early stage of the semester (e.g., SSH, Virtual machine, Vagrant, Docker) took students less time to solve than assignments from a later stage of the semester (e.g., Ansible, Jenkins, Logstash). Looking at the figure, a visual comparison between the times recorded for tools that both universities have in common suggests that TUB students were slightly faster solving Docker, Kubernetes and Logstash assignments, but took significantly less time solving the Jenkins assignment. Using the Kolmogorov–Smirnov test on the data from each of the four assignments, we received p-values ranging from 0.39-0.94

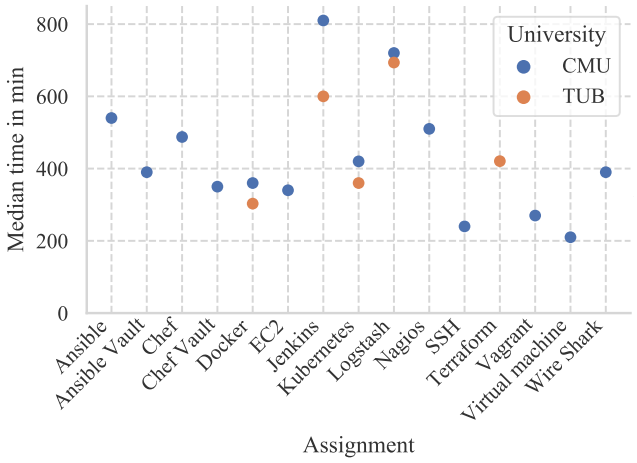

Figure 1. Median time over all data points from all semesters, grouped by assignment

which strongly indicates the null-hypothesis (samples drawn from populations with the same distribution) can be accepted, so that we assume the data sets are comparable. The Mood's median test provided p-values of 0.63 (Docker), 0.64 (Kubernetes), 0.02 (Jenkins), and 1.0 (Logstash), which suggests that students of both universities were similarly fast solving Docker, Kubernetes and Logstash assignments, but TUB students were faster solving the Jenkins assignment. When looking at assignments individually, we can also observe that Vagrant, Nagios, Logstash saw a time increase over the years (not shown in the figure). Contrary to that trend, for SSH, Docker, and Kubernetes assignments students expended a stable or slightly decreasing amount of time over the years.

4 Lessons learned and experiences

As instructors of the courses, we are confronted with a challenge: DevOps is a quickly shifting field and particularly its heavy dependence on automation tools gives the field constant momentum. While some of the tools are established by now, new solutions emerge frequently and influence numerous traditional practices immediately. Our teaching goals for the course are thus divided into two sub-goals which are reflected in the lecture and assignment content. *1) In the lecture* we focused on teaching underlying, lasting concepts of software engineering, including but not limited to threshold concepts [5], and core concepts of DevOps in particular, and chose inverted classroom questions that we thought would help deepen students' understanding of the subject. *2) In the assignments* we focused on exposing students to essential techniques and more recent trends and tools (e.g., Terraform: v1.0.0 release, 8 June 2021[10], assignment started 23 June 2021).

[10]https://github.com/hashicorp/terraform/releases/tag/v1.0.0, accessed 2021-07-16

Lecture. The content and inverted classroom format of the lecture are well received by the students – student evaluations of the course at both universities included the sentence "best I have ever taken/had". This was also reflected in the overall rating, which was well above average. However, some students saw the repetition of content from the video lectures through discussions in the classroom as repetitive and little helpful. Additionally, at German universities participation is not supposed to be mandatory, which inherently conflicts with the inverted classroom style. In-class discussion is arguably essential to foster deeper understanding in inverted classroom settings. At TUB, we mitigated the issue by offering a small percentage of the points for the whole course for in-class participation, with an easy-to-meet bar for achieving full points in terms of quality of answers or comments from the students.

While many discussion questions focused on deepening the content from the video lectures, some discussion questions were targeted at applying the knowledge and asked about implications of certain statements. For instance, in one lecture video, the Amazon design rules [11] were presented; the discussion question then asked about the implication of designing services from the ground up to be externalizable. After some deliberation and discussion, students mentioned among others, that API design had to play a bigger role, and security considerations had to be implemented from the start of the development process.

When the COVID pandemic started in 2020, the inverted classroom style facilitated transitioning to fully remote teaching. Since the lectures were already recorded on video, the pre-class activities were unchanged. In-class activites, however, changed significantly. On the technical side, some students do not have cameras, or sometimes the technology fails, which posed interruptions to the course.

At CMU, the in-person discussion also requires the student to pay attention at all times because they might be called on to continue a line of questioning. Students' body language reveals to the lecturer how much they are paying attention. The ones that were not paying attention became more likely to be the next called on. With remote teaching, the ability to measure engagement is much reduced. Many times, the response of a student to a question is "could you repeat the question". This indicates the students answering in this way were not paying attention.

During the in-class lectures at TUB we observed that more lively discussions and a more productive atmosphere developed when students did not feel pressured by our questions and were able to contribute only when they felt they actually had points to contribute. Still, we wanted to provide equal chances to address the class to all students, while ensuring that all students also participate actively. To achieve this, we asked students to raise their hand (virtually in the video conferencing tool) if they wanted to contribute and invited them to speak when it was their turn – while keeping track of the number of inputs per student. If the number of contributions became imbalanced, we counter-acted by skipping students' raised hands or announcing that we expect contributions from certain students and calling on them shortly after (usually from a raised hand).

In addition to the usual inverted classroom blocks, we also had guest lectures at TUB. During those were able to observe that the discussions were more stimulating when the lecture content was provided as a video ahead of the time and the guest had a conversation with the students in an inverted format – even if the guest's content was watched only briefly right before the inverted classroom started.

Assignments. We asked the students to record how much time they spend on each assignment. This allows us to announce expected time expenditure to make it less frustrating for students, simplifying their multi-course scheduling problems.

A particularity of the assignments was that they were vaguely formulated and usually just stated a starting point for the assignment and the goal (e.g., deploy a mock application in a Docker container). We decided to use vague wording purposefully, to give students the chance to try out the tools themselves, without forcing them into a solution path. At TUB we took two lessons away from that: 1) vague wording comes to the cost of variance in the formal quality of the submissions, and 2) it is important to manage expectations accordingly. Students were typically more used to clearly defined solution spaces and complained about the additional freedom, unless we clearly stated that to be a purposeful design.

Post class experience. From guest lectures and feedback we received from students after they found jobs, we learned that the lecture content is well aligned with industrial software engineering practice.

Multiple students have reported after working in their jobs for a while that they found the DevOps course to have been of great value. Some students moved directly into a DevOps role or DevOps trainee role. At TUB, many students had jobs parallel to their studies and gave similar feedback.

Students also reported that their interview experiences were different from those of their colleagues who did not take the course. The standard industrial interview basically questions the student on their algorithmic knowledge. Some of our former students reported that they were sent on a different interview path to explore their knowledge of DevOps.

5 Discussion and Conclusions

The analysis in Section 3 is subject to threats to validity. First, self-reported times are not particularly reliable. The

[11] https://presentationtube.com/watch/?v=vZRsbfnIeqV

representativeness of the data can not be guaranteed, among others for two reasons. First, at TUB, we made clear in the kickoff meeting that the course will likely be perceived as challenging, so some students might have not applied for participation. Second, since participation in our study was of course optional, not all students participated, and hence those did not hand in their time sheets. The lessons reported in Section 4 are subjective, and may or may not be replicable in other contexts.

In summary, we found that the inverted format worked well in a remote setting. Students valued the style of teaching and content, and some employers valued the resulting skill sets. Given the open accessibility of the video content, other universities could offer similar modules with relative ease. However, teaching DevOps poses additional tasks for the instructors since the field and tool landscape changes rapidly.

References

[1] Len Bass and John Klein. 2019. *Deployment and Operations for Software Engineers.* Amazon.

[2] Len Bass, Ingo Weber, and Liming Zhu. 2015. *DevOps: A Software Architect's Perspective.* Addison-Wesley Professional. https://doi.org/10.1007/978-0-134-04984-7

[3] J.L. Bishop and Matthew Verleger. 2013. The flipped classroom: A survey of the research. *ASEE Annual Conference and Exposition, Conference Proceedings* (01 2013).

[4] Chris Hughes, Sue Toohey, and Sue Hatherly. 1992. Developing learning-centred trainers and tutors, Studies in Continuing Education. *Studies in Continuing Education* 14, 1 (1992), 14–27. https://doi.org/10.1080/0158037920140102

[5] J. H. F. Meyer, R. Land, and C. Rust. 2003. Threshold concepts and troublesome knowledge: linkages to ways of thinking and practising within the disciplines. In *International Symposium on Improving Student Learning*. 412–424.