Theme Article: Special Issue on Software Engineering Education & Training

Live is Life—Teaching Software Engineering on Live Systems

Rolf-Helge Pfeiffer, *IT University of Copenhagen, Denmark*
Mircea Lungu, *IT University of Copenhagen, Denmark*
Paolo Tell, *IT University of Copenhagen, Denmark*

Abstract—The majority of resources in software engineering are spent on maintenance and improvement of existing software. However, academic software engineering education usually focuses heavily on tasks with lower consumption of resources in practice, e.g., requirements engineering, system modeling, green-field engineering, etc. In this article, we present the course "DevOps, Software Evolution and Software Maintenance" *that we developed and teach at IT University of Copenhagen. It differs from similar courses in the field in that it focuses student project work on maintenance and evolution of a live software system that is under simulated load for two-thirds of the course period. Our goal with this article is twofold: to encourage educators to incorporate the aspect of evolution and maintenance of live systems into software engineering curricula; and, to encourage practitioners to share cases of systems with captured real-world workloads so that they can be integrated in simulations in education.*

We perceive a mismatch between how software engineering (SE) is often taught in higher education and how it is applied in industry.

A significant amount of contemporary systems are not newly developed but evolve over time while being used live in production and often the largest amount of resources in SE projects is spent on maintenance (60% according to [1], 75% according to [2], and 80% based on [3]). Contrarily, academic SE courses are often structured around books like Sommerville's *Software Engineering* [4] or Bruegge's *Object-Oriented Software Engineering* [5] with projects focusing on green-field development stressing requirements engineering, system modeling, software design with respective notations like UML, etc.

Other academics like Spinellis perceive this mismatch too: *"[SE education] often focuses on how to single-handedly develop programs from scratch in a single language and single execution environment, a development style prevalent in the 1950s and 60s. Nowadays, software development is typically a team-based activity and most often involves extending and maintaining existing systems written in a multitude of languages for diverse execution environments"* [6]. And, more pragmatically, practitioners describe the mismatch: *"in university, they teach you how to write a 400-line program that solves a problem from A-Z. You have a blank canvas, and you need to show off your knowledge [. . .] In the end, you have a nice solution to a straightforward problem [. . .] Professional software developers work in groups [. . .] and more often than not – it's fixing stuff rather than building it from scratch."*[1]

SE education not matching industry requirements is against public interest. For example, the Danish Government formulates the goal of higher education as follows: *"it is paramount, that we provide relevant and high-quality educations that match industries requirements."*[2]

To increase practical relevance, educators are

[1] https://vadimkravcenko.com/shorts/things-they-didnt-teach-you/
[2] https://www.regeringen.dk/nyheder/2018/maalsaetninger-for-de-videregaaende-uddannelser/

XXXX-XXX © 2023 IEEE
Digital Object Identifier 10.1109/XXX.0000.0000000

switching towards project-based teaching covering more realistic aspects of SE. Some ask students to contribute to open-source software[3,4][7], while others focus on industrial collaboration when teaching technologies and practices related to DevOps [8]. Some teach software evolution and maintenance already in introductory SE courses [9], others focus on deliverability and deployability (CI/CD pipelines) [10], and yet others focus on DevOps skills and related cloud technologies [11], [12].

However, none of the courses mentioned above, none of the ones identified in a recent meta-study [13], and, to the best of our knowledge, no other academic SE course include the aspects of maintenance and evolution of **live** software systems – i.e., systems that are in production and have to be updated and scaled while being in use. The importance of this aspect of modern software development is proven by the adoption of DevOps practices in industry [14].

Therefore, we have developed a project-based course called *"DevOps, Software Evolution and Software Maintenance"*, which we teach at IT University of Copenhagen every spring since 2020. During this course, students take over a legacy software system running in a legacy production environment. It is a Twitter-like minimal micro-blogging web-application. Throughout the entire term, students evolve and maintain it. What differentiates our course from other academic SE courses is that we utilize a simulator for two-thirds of the semester, which interacts with the students' systems. This simulator registers users, allows them to tweet, follow each other, and performs other actions. The simulator gradually increases load on the students' systems. Thereby, it serves as a catalyst for software evolution and maintenance tasks.

When creating the course, our goal was to design a subject covering the software life-cycle phases that we believe are underrepresented in more "traditional" academic courses. Based on our experiences in industrial SE, we wanted to let students experience practice-oriented SE work in a safe environment while developing, maintaining, and evolving a 24/7 system.

This article is relevant for educators or trainers since we describe the course design and discuss challenges when teaching it. Practitioners get out of this article how to support education of future colleagues by sharing software together with usage data. Recruiters, project managers, or team leaders, can use this article to understand the skill set of graduates that completed this course and that their project work can be used for assessment of candidates in hiring procedures.

COURSE DESCRIPTION

The course is setup around the case of a Twitter-like minimal micro-blogging web-application, which we call *ITU-MiniTwit*. The "background story" presented in the first lecture is that in 2012 former students developed a minimum-viable-prototype of ITU-MiniTwit directly in a production environment without any documentation or following any modern SE practices. Students (in groups of usually five) take over, evolve, maintain, and scale ITU-MiniTwit throughout the course (we call these the *student systems*).

After several initial sessions, each group deploys and operates their student system live in production on publicly accessible Virtual Private Servers (VPSs). Eventually, an automated program (called the *simulator*) simulates users interacting with the student systems, i.e., users registering, tweeting, following, etc.

All course material is hosted online on GitHub.[5] Each lecture introduces one or more topics, which we typically illustrate or demonstrate on a prototypical case with an exemplary technology. For example, we demonstrate packaging and deployment of applications or components via Docker containers, deployment to VPS on DigitalOcean, monitoring with Prometheus and Grafana, cluster management via Docker Swarm Mode, etc. However, students are free to choose alternate technologies, programming languages, frameworks, tools, etc. if they can justify their choices.

Figure 1 illustrates the course setup. The topics of the 14 lectures are listed on the left with corresponding topics of project work on the right. Small black squares illustrate weekly lecture and exercise sessions. The gray frame in the center illustrates the period in which the student systems are live and under load of the simulator. The plot illustrates the increasing number of requests that the simulator sends to the students' systems.

In the first session of the course, student groups, take over a legacy version of ITU-MiniTwit – a Python 2 application based on Armin Ronacher's original *MiniTwit* application which used to be an example application for the Python web-application framework

[3] https://avandeursen.com/2013/12/30/teaching-software-architecture-with-github/

[4] https://gist.github.com/ruimaranhao/b2c64e906ac9a6bcad02

[5] https://github.com/itu-devops/

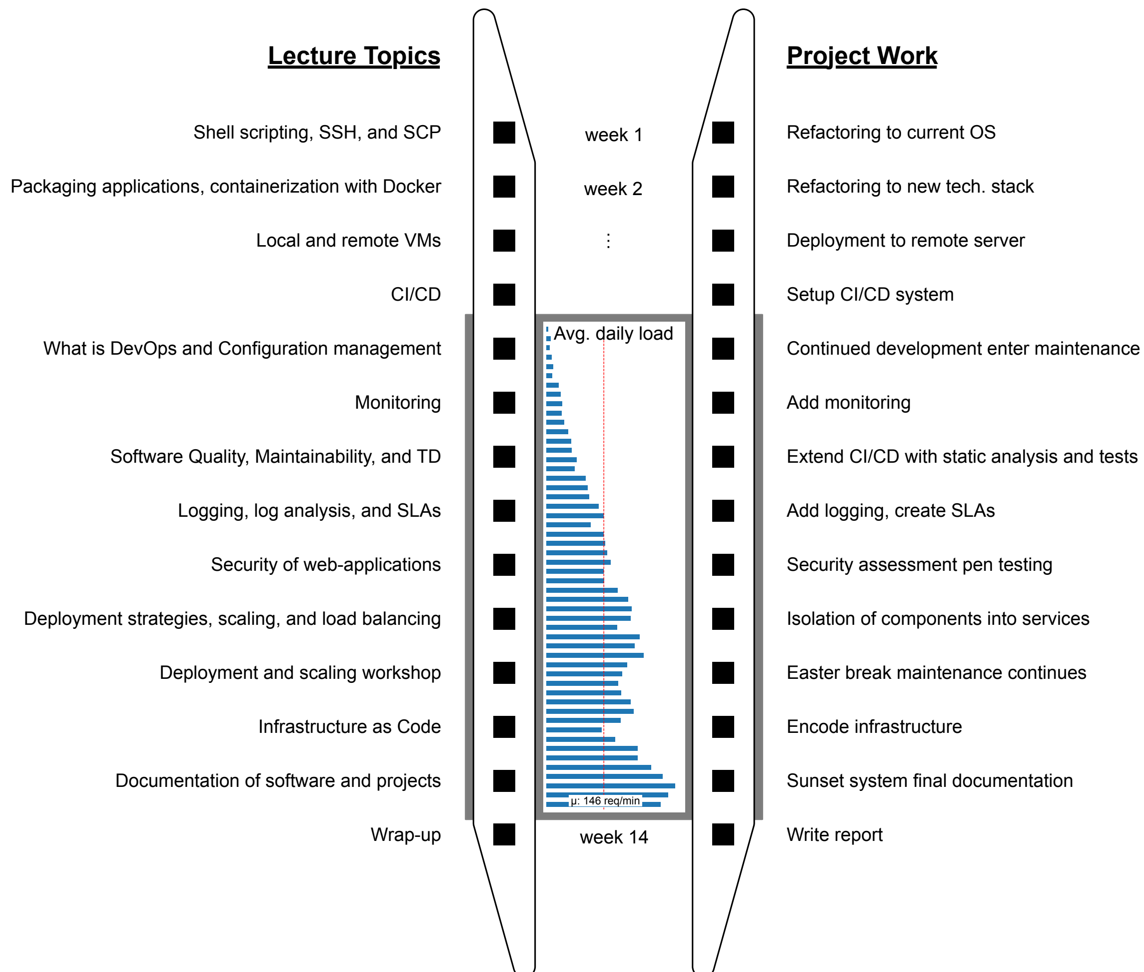

FIGURE 1: Overview of the 14 weeks elective course (7.5ECTS) with one combined four-hour teaching and practice session per week (black squares). Lecture topics are on the left and project work topics on the right. Centered is an illustration of the daily number of requests that the simulator sends to student systems.

Flask.[6] Here, "take over" means that all artifacts that form the application are manually copied from a remote legacy server running a deprecated Ubuntu OS (12.04) with outdated dependencies. After taking over the application, student groups have to put all these files under version control (Git repository on GitHub) and refactor it so that it can be executed on a current Ubuntu Linux with Python 3, i.e., an exercise in *adaptive maintenance*. All subsequent tasks of project work listed in Figure 1 are executed in these repositories and on the continuously refactored and evolving artifacts that groups took over originally.

Our university does not offer a computing lab with homogeneous environments for the students to work on. To ease development in heterogeneous environments, already the second session introduces packing and executing applications in Docker containers. Once creation of VPS is introduced (session three), the production environment for manual or locally scripted deployments are remote servers.

In accordance with principles of DevOps and agile development, we encourage students to work and deploy continuously. To make work visible (one of the DevOps principles [14]), we provide dashboards

[6] https://github.com/pallets/flask/tree/1592c53a664c82d9badac81fa0104af226cce5a7/examples/minitwit

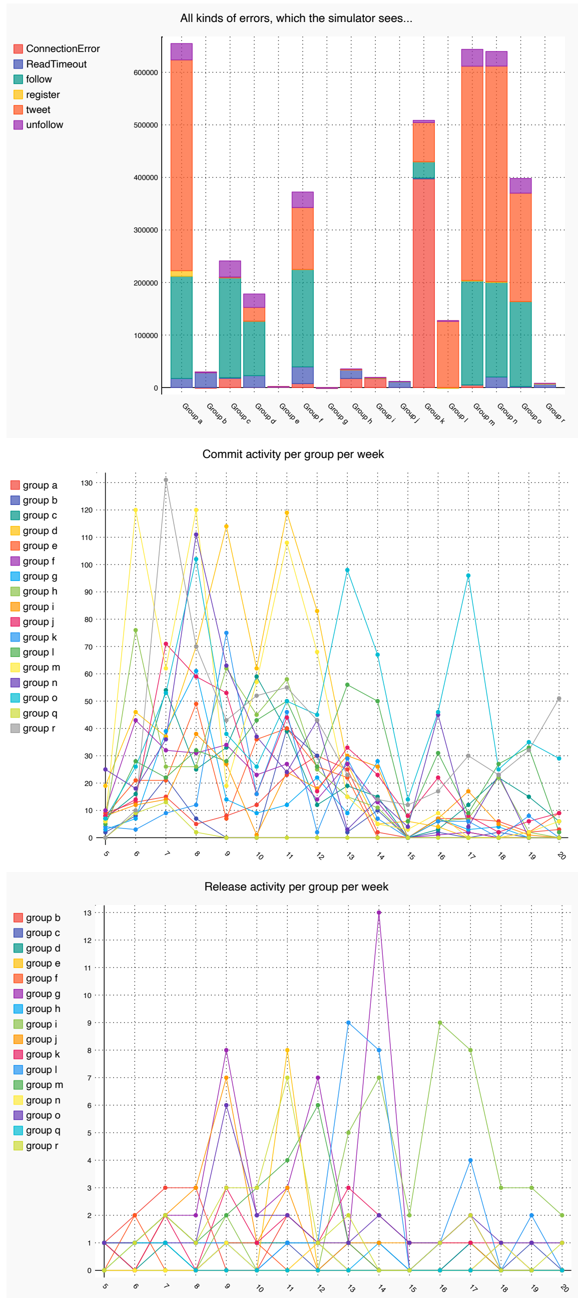

FIGURE 2: Dashboards visualizing: (top) types of errors that the simulator received from student systems within the first two weeks in 2022; (center) weekly number of commits per group; and, (bottom) number of releases per week per group. All dashboards are from the course in 2022.

that visualize, e.g., the number of weekly commits per group or the number of weekly releases per group, see bottom of Figure 2. By week four, deployments happen automatically via CI/CD chains created in project work.

From week five on, all student systems are considered to be live and "in production". For the next weeks, the simulator feeds events like persons registering, tweeting, following each other, etc. to student systems. To make that possible, students have to implement a web-API for which they receive a specification and an automated API test in week three. All subsequent project work takes place while simulated users interact with student systems. Note, besides simulated user interactions, all student systems are publicly accessible on the internet. Anyone can register and send messages during the live period on any student system. Moreover, student groups interact with other group's systems to investigate feature completeness or quality.

Figure 1 illustrates, that frequency of simulator requests increases over time, which mimics a situation in which student systems become increasingly popular. The timed behavior of simulator request frequencies is based on the database dump of HackerNews[7] and compressed to fit over 10 million requests into the time frame of the course.

Together with sending user requests, the simulator monitors responses that do not conform to its web-API specification. The top of Figure 2 illustrates one of the views of the course dashboards. Per group, it illustrates the sum of all errors that the simulator observed while interacting with the student systems during weeks five and six in 2022. For example, it shows that the systems of groups a, f, m, n, and o appear to respond not as specified when receiving tweets and follow requests or that Group k's system was down (ConnectionError). These dashboards illustrate potential issues and enable self-reflection in groups. Based on them, we start each teaching session discussing statuses of student systems and solutions to potential problems.

During the first weeks of live operation, students have limited possibilities to detect and reason about issues or security incidents. Once monitoring and logging are setup and configured (weeks six and eight respectively) students experience that the quality of logged and monitored information has a direct influence on their ability to detect and reason about potential issues or incidents. Typically, the logging and monitoring improve over time, reflecting the lessons from past shortcomings.

As the course progresses, load from the simula-

[7] https://news.ycombinator.com/

tor increases, which motivates addressing basic techniques for scaling in weeks 10 and 11.

At the end of the course, after session 13, we stop the simulator. This allows students to focus on wrapping up, writing reports, and prevents them from having unnecessary spending. Indeed, many of the services that we rely on in this course would not normally be free. However, we have selected services that offer free tiers for students (e.g., the GitHub Student Developer Pack[8] offers students a large amount of free credits, including DigitalOcean).

EVALUATION

The popularity of the course has increased from about 50 enrollments (2020) to roughly 80 (2022), a large number for an elective course at our university.

In 2020 and 2022, we evaluated the course with the Delphi method[9] [15], collecting up to three positive and three negative statements per student. After data cleaning, we receive 246 comments (2020: 148; 2022: 98). We clustered responses via a light thematic analysis and identified several recurring themes, which we present with quotes from evaluations below.

Relevance and value. The course and the topics covered are particularly appreciated as they are *"very relevant and useful [...] in the 'real world"'*; *"extremely relevant for me as developer"*; and, *"[an] excellent overview [of] what devops is and which tasks are waiting for one who proceeds in this career"*. More than 16% of the positive statements concern the relevancy of the course, and the above are just a sample of these consistent comments. Additionally, students value the use of modern technologies. One says that the course *"feels very relevant. I am currently also working with Docker, Prometheus, Grafana, etc. in my part-time job!"*; while another highlights that the course *"introduced many valuable tools that can be used in my future career"*.

Hands-on and practical. The experience of working on a live system, is continuously appreciated. Students refer to it as a *"great [...] hands-on course, instead of just theory"*; emphasizing how *"the hands-on approach of taking over an old system and revamping it with new DevOps concepts worked really well for learning"*; and recognizing that it is *"nice how practical the course was, getting our hands dirty is a lot more fun than watching someone else do it"*.

Fun and freedom. Besides being relevant and providing professional value, students appreciate the course as *"super fun class"* and *"fun project"*. Others state that: *"the simulator was fun because it felt like real users"* Finally, students value *"[...] freedom in terms of choosing your tech stack"* and consider it *"[...] awesome that you [are] allowed for experimentation"*.

Workload and time pressure. Naturally, the course is also very demanding. One states that it is *"[...] the 7.5 ECTS course I have spent the most time on since I started"*, while another mentions how *"the workload can be quite large which can lead to prioritizing other courses"*. Some students consider the perceived support insufficient and problematic due to the same freedom considered positive by others: *"everyone having different systems meant you could not really get help with any technical problems"*.

OBSERVATIONS & REFLECTIONS

An Industry-relevant Course

When planning the course, before its first iteration in 2020, we discussed the syllabus with two consultants from a consultancy[10] focusing on DevOps and continuous delivery. Furthermore, we informally gather feedback from our industrial guest lecturers on the relevancy of the course each year. So far, we discussed the course with seven guest lecturers who work as consultants for Eficode, Accenture, and the Implement Consulting Group. They value that it is one of the few courses in the country that teaches DevOps concepts and related technologies, while – importantly – allowing students to experience software engineering and the effects of their decisions in a more realistic environment.

This course does not only decrease the gap between industrial practice and academic education, it also makes students more effective in subsequent courses and projects as it increases the quality in their production of software. For example, we know that students that took this course establish automatic CI/CD chains in proceeding courses to automate builds, tests, and static analysis thereby increasing productivity and quality of their projects.

A Safe Place to Practice

Students learn and gather experience in a realistic but safe manner in which even disastrous events like complete loss of user data can occur and serve as a learning experience.

[8] https://education.github.com/experiences/intro_to_web_dev
[9] https://github.com/HelgeCPH/delphi_evaluator
[10] https://www.eficode.com/praqma

For example, in multiple iterations of the course, some groups had their databases (those storing user information, tweets etc.) encrypted and held to ransom due to exposure of vulnerable default configurations. Similarly, multiple groups completely lost data stored in databases either due to inappropriate storage of data within ephemeral containers or due to erroneous overwriting during updates of the live system, e.g., when setting up automatic deployments.

Even though such experiences are unpleasant to the respective groups and may cause frictions in project work, they result in important class conversation followed by the application of mitigation strategies, e.g., backups, increased security hardening. Furthermore, they serve as learning experiences that trigger – as we believe – long-lasting changes in the student awareness.

Promoting System Thinking

Every year, students become aware of the impact of their design decisions on their system performance. A common issue groups face is the modeling of data based on premature assumptions, e.g., which data is important and should be supported by database indexes. Over time, students learn that the "real-world" used by the simulator differs from initial assumptions and the performance of their systems gradually decreases.

Similarly, even though likely obvious to experienced professionals, students experience the effect of hosting their systems in data centers on the other side of the globe, which generally results in response times that are considered "unacceptable" by the simulator. This triggers for instance the necessity to move systems to other data centers or cloud vendors, which in turn exposes directly the quality of encapsulation of system components and their deployability.

Fostering Cost Awareness

Being responsible for evolving and maintaining a system in a live environment directly highlights the costs associated with software. Depending on the design and resources required of student systems, groups might run out of free credits, e.g., on DigitalOcean, which triggers saving strategies to share costs among accounts.

For example, groups that deploy their systems as loosely coupled Docker containers and encode infrastructure as code usually experience that they can move their systems with no or minimal modifications even between cloud vendors. Also, groups that specify their infrastructure as code save costs by "hibernating" their systems until demonstrations in the exams.

The effect of design decisions to operational cost cannot be experienced by discussing software quality in terms of design patterns, architectural properties, or code quality.

A Space for Varying Skills and Motivations

We observe that some groups find it more challenging than others to cope with the project workload and the number of new concepts presented weekly. However, our simulation has proven to be modest enough to be handled with various levels of technical skills and motivation.

For example, we see all kinds of student systems ranging from simple web-applications that are deployed on the smallest available VPSs with possibly a load balancer in front[11] to applications that are deployed as services onto self-managed Kubernetes clusters.

During group formation, we recommend students to discuss motivation and skills to allow everybody to have a fun and instructive experience.

CHALLENGES

Being responsible for a hands-on course with simulated live environment is challenging. In this section, we focus on educators and trainers and discuss what to be mindful of when teaching a similar course. We focus on what is required on top of executing "regular" lectures and exams.

Developing a Live Simulator

Developing simulations is difficult and time consuming. Our simulation is an artificially generated API load test spanning ten weeks based on a scenario in which hypothetical users register (user names generated using US census data), send hypothetical tweets (simple language model trained on all Sherlock Holmes books), un-/follow each other randomly, and in which the frequency of actions increases over time. Writing code that generates such a simulation is time consuming[12] and it is case-specific.

[11] https://www.digitalocean.com/community/tutorials/how-to-create-a-high-availability-setup-with-heartbeat-and-reserved-ips-on-ubuntu-16-04

[12] We are happy to share the simulator upon request. However, it is not shared publicly such that it does not influence the students' solutions.

Developing (Legacy) Case Systems

Besides the initial legacy version of ITU-MiniTwit that students take over in the beginning, case-based learning requires cases for sessions that illustrate respective topics. Currently, we have eight repositories under the course organization on GitHub[13] with adapted cases that illustrate, e.g., how to containerize ITU-MiniTwit, how to deploy it on VPSs, or how to add monitoring. Developing and maintaining such cases takes time and is an investment, which makes changing cases in future harder.

Additionally, developing legacy systems like the initial version of ITU-MiniTwit, i.e., software that looks as if written more than a decade ago, is actually quite hard. Amongst others, package repositories of end-of-life Linux distributions are offline and deprecated versions of library dependencies have to be unearthed.

Adapting to Class Size

Scaling the class up to 80 students requires teachers to carefully assess resources before semester start.

Our teaching assistants and sometimes we ourselves meet the student groups weekly and discuss project statuses.

However, not only the human resources have to be scaled, but also the computational resources as a failure in the simulator would break the students' learning experience. Consequently, we run a pre-simulation prior to the actual simulation with dummy student systems just to assure that our infrastructure is capable to handle the actual scenario. This pre-simulation requires time for organization, execution, monitoring, and validation.

Maintaining a Simulated Live Environment

As teachers, we are responsible for keeping a 24/7 simulator operational next to preparing and running lectures. Usually this does not consume excessive time during the term, but requires time before each new iteration of the course, e.g., VPSs have to be instantiated. We encoded the simulator setup via Vagrantfiles, Bash, and Python scripts. However, regular software maintenance tasks like update of dependencies and environments to recent versions have to be performed. Such seemingly small changes can have unforeseen consequences. For example, we experienced in 2021 that a combination of Python interpreter with a library caused a memory leak, which broke the simulator due to excessive memory consumption. This lead to late night debugging sessions to circumvent the issue.

[13] https://github.com/itu-devops

Targeting Heterogeneous Environments

Most students work using their laptops, which have heterogeneous hardware, which can make sessions that require virtualization software like Docker or VirtualBox challenging.

From a teaching perspective, we decided to tackle that challenge by targeting a single environment (most current Ubuntu LTS version). We encourage students to setup a corresponding environment on their computers. This is an imperfect solution from the students' perspective since certain computers, like the new ARM Macs cannot yet host it natively.

RECOMMENDATIONS & CONCLUSION

Based on our experiences with developing and teaching a course that incorporates operation and maintenance of live software, we have three main recommendations for various roles in the SE community:

For educators, trainers. To adequately prepare students for their professional careers, we recommend that evolution and maintenance of live systems should be integrated into course work. In case no real-world usage data is available, simulated usage data that puts load on systems can be used to facilitate teaching.

For practitioners. To contribute to adequate education of future software engineers, we recommend software systems are shared publicly together with representative usage data. Finding suitable case systems that can be used in teaching is quite difficult. Finding respective usage data, i.e., how users interacted with live systems that can be used to "replay" certain scenarios is virtually impossible.

Additionally, project based courses with live systems like ours allow for involvement from industry, e.g., in roles like advisors, assessors, or by providing skills demanded in real-world DevOps.

For recruiters, project managers, team leaders. When hiring new employees, recruiters, project managers, team leaders, etc., can use a candidate's project that was developed in this course to gauge a candidate's capabilities. That is, besides grades or certificates, the ability to reinstantiate a live system and it's demonstration can form a basis for assessment or a case in job interviews.

In this article, we describe the design of the course *"DevOps, Software Evolution and Software Maintenance"* that we developed and teach at IT University of

Copenhagen. The course is different from similar SE courses in that it focuses project work on maintenance and evolution of a live software system that is under simulated load for two-thirds of the course period.

Students evaluate this course to be a fun and challenging experience, practitioners consider the skills that students acquire to be relevant.

With this article, we aim to encourage educators to incorporate the aspect of evolution and maintenance of live systems into SE curricula, and we aim to encourage practitioners to share cases of systems with captured real-world workloads to allow their integration in education.

ACKNOWLEDGMENTS

We thank Jens Egholm Pedersen for his contributions to earlier iterations of this course, especially on the sessions on logging and monitoring, and we thank all our teaching assistants for their valuable contributions.

REFERENCES

1. R. L. Glass, "Frequently forgotten fundamental facts about software engineering," *IEEE software*, vol. 18, no. 3, pp. 112–111, 2001.
2. R. Stephens, *Beginning Software Engineering*. Wiley, 2022.
3. T. M. Pigoski, *Practical software maintenance: best practices for managing your software investment*. Wiley Publishing, 1996.
4. I. Sommerville, *Software Engineering*. Pearson Education, 2016.
5. B. Bruegge and A. Dutoit, *Object-Oriented Software Engineering Using UML, Patterns, and Java: Pearson New International Edition*. Pearson Education, 2013.
6. D. Spinellis, "Reading, writing, and code: The key to writing readable code is developing good coding style.," *Queue*, vol. 1, no. 7, pp. 84–89, 2003.
7. J. Buchta, M. Petrenko, D. Poshyvanyk, and V. Rajlich, "Teaching evolution of open-source projects in software engineering courses," in *2006 22nd IEEE International Conference on Software Maintenance*, pp. 136–144, IEEE, 2006.
8. 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, IEEE, 2019.
9. V. Rajlich, "Software evolution and maintenance," in *Future of Software Engineering Proceedings*, pp. 133–144, 2014.
10. A. Capozucca, N. Guelfi, and B. Ries, "Design of a (yet another?) devops course," in *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment: First International Workshop, DEVOPS 2018, Chateau de Villebrumier, France, March 5-6, 2018, Revised Selected Papers 1*, pp. 1–18, Springer, 2019.
11. H. B. Christensen, "Teaching devops and cloud computing using a cognitive apprenticeship and storytelling approach," in *Proceedings of the 2016 ACM conference on innovation and technology in computer science education*, pp. 174–179, 2016.
12. I. Alves and C. Rocha, "Qualifying software engineers undergraduates in devops-challenges of introducing technical and non-technical concepts in a project-oriented course," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pp. 144–153, IEEE, 2021.
13. M. Kuhrmann, J. Nakatumba-Nabende, R.-H. Pfeiffer, P. Tell, J. Klünder, T. Conte, S. G. MacDonell, and R. Hebig, "Walking through the method zoo: does higher education really meet software industry demands?," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pp. 1–11, IEEE, 2019.
14. G. Kim, J. Humble, P. Debois, J. Willis, and N. Forsgren, *The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations*. IT Revolution, 2021.
15. C. M. Goodman, "The delphi technique: a critique," *Journal of advanced nursing*, vol. 12, no. 6, pp. 729–734, 1987.

Rolf-Helge Pfeiffer is an Assistant Professor at the IT University of Copenhagen in the Research Center for Government IT. His research interests and teaching are in the areas of software engineering, software quality, software quality metrics, and technical debt. Prior to his academic career, he worked as a software engineer and team lead at the Danish

Meteorological Institute, where he was responsible for development and maintenance of 24/7 remote sensing software systems.

Mircea Lungu is an Associate Professor in computer science at the IT University of Copenhagen. His research interests and his teaching are focused on software visualization, tools for the analysis and steering of software evolution, and human computer interaction, and personalized learning environments. He is the main author and maintainer of the API for zeeguu.org, a research system live since 2018 that supports language learners in studying text and vocabulary in a hyper-personalized manner.

Paolo Tell is an Associate Professor of Software Engineering at the IT University of Copenhagen's Computer Science department, Denmark. His research and teaching are centered on industry-academia collaborations. His research foci include software processes and software process improvement, teamwork, computer-supported cooperative work, and (globally) distributed software engineering. Prior to his academic carrer, he worked, among others, as software engineer involved in the development of an aircraft simulator.