Threading DevOps Practices Through a University Software Engineering Programme

Robert Chatley
Dept of Computing
Imperial College London
rbc@imperial.ac.uk

Ivan Procaccini
Dept of Computing
Imperial College London
ip914@imperial.ac.uk

***Abstract*—In preparing students for a future career as software engineers, in addition to learning the fundamentals of Computer Science, we want them to develop a set of practical skills and professional ways of working. Current industry trends are towards the adoption of a *DevOps* culture, where software engineers are responsible not only for the design and development of software systems, but also for their deployment and operation in production. In this paper, we describe how we have introduced students to DevOps practices early in their degree programme, and then woven their use through many different projects as an underpinning thread. Rather than being taught in isolation, these professional practical skills are practised repeatedly through the degree, in different contexts, to the point where they almost become second nature.**

We describe concretely the structure of our introductory *DevOps Lab* exercise and how it is assessed, explain how DevOps skills are applied in later projects, and report on our experience so far of teaching these techniques in this manner.

***Index Terms*—curriculum design; devops; software engineering education**

I. Introduction

Software Engineering curricula have to move with the times to keep up with an ever-changing discipline and the demands of industry. Over recent years it has become commonplace for university Computer Science and Software Engineering programmes to teach agile development methods [1]–[3]. These methods focus on iterative product development, with particularly Scrum and eXtreme Programming focussing on cyclical development in fixed length iterations. As a development approach this works well, but industrial teams have seen a limitation in these types of methods in recent years. In focussing on the software development, the design and construction of the system, we are missing out on the delivery of the software into the hands of the customer or user – the so-called "last mile" [4].

In industry, the traditional separation between software developers who produce software and the operations teams that deploy and run the software in production has been narrowed as teams work towards a *DevOps* culture. These disciplines are brought together to form a single team of engineers responsible for the design, development and delivery of a software system [5], [6]. Rather than being just a software development team, we now have a product team, and this requires engineers to have a wider range of skills. As well as being able to refine requirements, write code and create test suites, they now need to be familiar with (amongst other things) configuring build pipelines, provisioning cloud resources, and deploying software into production.

Rather than being a specialist activity undertaken by a specialist team, disciplined operational practices are becoming core to every successful development team. Recent studies show that adoption of continuous delivery and DevOps practices is aligned with high performing technology organisations and business success [7].

As the technical practices associated with DevOps became commonplace, we wanted to integrate them into our university curriculum. But, rather than introducing them in a dedicated module, we have woven a thread through several different modules and projects, emphasising the underpinning nature of these practices. Wherever a software development project is undertaken as part of the course, particularly where it is tackled as a team, we want the students to be able to draw on appropriate technical practices to support them in producing high quality work. We want students to really feel the benefit of putting these tools and practices into action, and to give them a set of tools that they can and will use in future projects, not because we told them to, but because they know they will help them produce better results.

For many years we have introduced students to version control on day one of their studies, after which they will use it almost every day during their degree for every exercise or project that involves working with code, in the same way that they would in a professional environment. The goal is not that we teach students about version control, but instead that they develop a habit of using it, and that this in turn supports all of their technical work. Similarly, we do not want the application of DevOps practices to be a goal in of itself, but rather a means to an end, supporting students in iteratively delivering complex projects in a professional manner.

The overall philosophy of our curriculum for Software Engineering is inspired by *lean learning* [8], where we transmit just enough knowledge for students to gain and apply skills in a practical setting, and then let them develop those skills through substantial projects. In this paper, we describe in detail the format for teaching DevOps practices that we have developed in line with this philosophy, including details of the practical exercises, assessment techniques, and connections to later projects.

II. Background

A. DevOps

The term *DevOps* [9] represents the fusion of cultures, practices and tools belonging to software development and infrastructure operations teams.

Even with agile methods becoming the dominant way for teams to develop software [10], there was still often a divide between the "makers" of the code (*Dev*) and its "operators" supporting applications in production (*Ops*). There was a natural conflict between the two teams, as developers were often seen as optimising for rate of change, whereas operations were optimising for stability. Bringing the two teams together to cooperate on reliable, rapid delivery of software has proved effective and has led to the software engineer's role evolving into one that covers the complete software life-cycle, from design, to development, to production support [11].

To operate effectively in a DevOps environment, a software engineer must have an increased awareness of the tools and techniques required to reliably put code into production on a regular basis. They need to know how to:

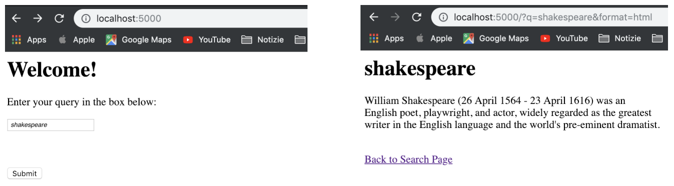
1) break down their implementation work into thin slices that can be integrated and deployed incrementally
2) design and implement a battery of automated tests for code to pass, giving confidence that each increment is of suitable quality to go into production
3) configure and maintain production environments for their application to run in (for example making use of cloud resources)
4) select and configure tooling to orchestrate this process.

In a nutshell, the objective of DevOps is to minimise, for any change introduced in a piece of software, the time interval between committing that change to a version control system and that change being part of the software the end-user uses.

B. Continuous Delivery

Continuous Delivery (CD) is a practice closely associated with DevOps [4] and builds on the practice of Continuous Integration (CI). CI involves (a) frequent commits to a common code repository, (b) automated building and testing of the code, and (c) generation of reports on build failures. CD then adds the notion that every successful build should be a potential release candidate and should be deployable to a production environment "at the click of a button". A further extension to this idea is to completely automate the pipeline and deploy every change to production (as long as all the quality gates are passed) with no further human intervention. This more extreme version of the practice is known as Continuous *Deployment*.

The steps of a build pipeline are executed in succession – usually compile, then run unit tests, run integration tests, deploy to a staging environment, run acceptance tests, and finally deploy to the production environment – and, on failure of any step, the whole pipeline stops. The key objective of Continuous Delivery/Deployment is that of getting code changes (be they features, configuration updates or bug fixes) to the end-users in a quick, predictable and reliable fashion, and failing fast when problems are detected.



localhost:5000

Welcome!

Enter your query in the box below:

shakespeare

localhost:5000/?q=shakespeare&format=html

shakespeare

William Shakespeare (26 April 1564 - 23 April 1616) was an English poet, playwright, and actor, widely regarded as the greatest writer in the English language and the world's pre-eminent dramatist.

Back to Search Page

(a) Query for *shakespeare* (b) Result

Fig. 1: Sample interaction with the web app.

III. The DevOps Lab

Our DevOps laboratory exercise (hereafter referred to as *DevOps Lab*) is undertaken by all second year undergraduate Computing students approximately one third of the way through Year 2. By this stage the students are already well versed in writing code and developing systems, so here we want to complement these skills with practices focussing on infrastructure and operations. The core of this exercise revolves around setting up a pipeline for the building, testing and deployment of a simple Java web application. We provide the students with a git repository containing an initial version of the application, which has some complete but basic functionality. The details of the web application are unimportant, but we designed something that allowed a very simple and immediate form of user interaction: the user typing a string query into a "search box" and the application returning a suitable result (see Figure 1). We chose a programming language that our students were familiar with, and kept the application as a simple skeleton. This allows the students to iteratively add simple features, or enrich the user interface, with small code changes. During the exercise students often want to make a small change, and see that change reflected in the user interface in production so that they know that their new code has successfully deployed. We wanted this to be achievable without the need to learn a new language or a complex web framework. The focus of the exercise is on the build and deployment process, not on the application itself.

After an introductory lecture, students work at their own pace, following our exercise specification. We also give them pointers to the official documentation for the suggested tools and to additional tutorial resources that they can follow to dive deeper into particular areas of interest. We wanted to strike a balance between giving enough guidance so that all students could complete the basics of the task and not revealing too many details, so that they had to do some research on their own and develop their skills in reading and understanding technical documentation.

The task involves iterating on the given application to add a few features, and at the same time evolving their build and deployment infrastructure from something simple running on the local network to a more sophisticated system encompassing a range of cloud tools and supporting infrastructure. The following sections give a detailed overview of the aims, general organisation and assessment criteria of the *DevOps Lab*. The full exercise specification as provided to the students is available online for further reference[1].

A. Aims

The intended learning outcomes for the *DevOps* laboratory exercise are for the students to be able to:

- assemble their own build pipeline from scratch
- perform basic system administration tasks on Linux
- demonstrate continuous delivery practices, developing and deploying an application iteratively and incrementally
- investigate a variety of DevOps tools
- select appropriate tools and techniques to apply to future software development projects within their degree.

B. Structure

The exercise is divided into three phases:

1) Deploy to a virtual machine on an internal network: For phase one, the students are asked to set up a virtual machine on an internal cloud and a three-stage pipeline using GitLab CI[2] to build, test and deploy the application to this VM. This phase involves installing and configuring a GitLab Runner[3] to execute the pipeline jobs, triggering a Maven[4] build with the relevant configuration file, and writing a shell script to launch the application cleanly.

2) Improving availability and scaling with an external PaaS: The deployment environment used in phase one (i.e. the virtual machine in the departmental cloud) is located within the university's internal network, meaning that the deployed application is not publicly visible and has limited scaling potential. In this second phase students move their deployment to a public cloud provider, so that their application is visible on the public internet. To make this an easy step, we make use of Heroku[5], a popular Platform-as-a-Service provider. Students can create free accounts and deploy their applications easily by using some simple tools and adding these to their build pipeline. The suggested path for the students is to (a) install the Heroku CLI and use it to initialise a deployment area on the Heroku platform, (b) write the necessary Heroku configuration files to launch their application, and (c) update the pipeline to also deploy the application to Heroku whenever the previous pipeline stages pass. We recommend that the students retain the deployment to the VM as a separate stage, so as to have a staging environment as well as a production one.

Overall, these technologies together are representative of a tech-stack the students might have to deal with later on in a future career as software engineers, but did not present an overly steep learning curve as long as we provided appropriate guidance. The selection we made was based on what was simple to use but still representative of what might be used in an industry environment, and freely available (we did not want students to have to enter their credit card details to sign up to any services).

3) Packaging and portability with containerisation: We motivate the third phase by giving the students a new feature to implement. We ask for the application to be able to provide its results not only in HTML, but also in PDF. We suggest that this be done by calling out to an external tool – pandoc[6]. This library is not provided in the Heroku environment, and so we introduce the idea of packaging the application together with all of its dependencies in a container image, and deploying that to the cloud platform. The students are presented with *Docker*[7], and given auxiliary online materials to quickly gain a working knowledge of the tool. They are then in a position to write a simple `Dockerfile` for their application and its dependencies. They are finally guided to change their Heroku app configuration so as to be able to deploy their containerised application. If everything goes well, their production application should allow them to download PDFs of their query results.

C. Sys-Admin Tasks

The virtual machine that we provide to the students is in a clean state. We do not pre-install the tools they need. This means that they must perform some basic system administration tasks, including installing packages, creating users etc. Working in a VM environment offers a good way for students to familiarise themselves with these sorts of tasks, which they cannot normally perform on our locked-down lab PCs.

D. Extensions

Once they have worked through the three phases of the exercise, we encourage the students to explore the extensive set of features offered by GitLab pipelines, ranging from convenient configuration capabilities, to Shared Runners, to Google Kubernetes Engine[8] and Prometheus[9] integrations. We also suggest that they investigate other comparable CI/CD tools, like Jenkins[10] or Circle CI[11] and consider migrating part of their pipeline to use a different tool. This allows them to see that the basic process and principles they learnt are transferable, even when a different set of tools is chosen, and gives them confidence in assessing and selecting appropriate technology for future projects.

E. Logistics and Assessment

The DevOps lab takes two weeks to run in total, and runs alongside other modules that the students are taking. The laboratory exercise itself runs over one academic week (i.e. five days) and is supported by 6 hours of laboratory sessions distributed over three days, during which help is available from staff and teaching assistants. The students however are free to

[1]https://www.doc.ic.ac.uk/ ip914/teaching/devops-spec.pdf
[2]https://about.gitlab.com/
[3]https://docs.gitlab.com/runner/
[4]https://maven.apache.org/
[5]https://www.heroku.com/what
[6]https://pandoc.org/
[7]https://www.docker.com/resources/what-container
[8]https://cloud.google.com/kubernetes-engine/
[9]https://prometheus.io/
[10]https://jenkins.io/
[11]https://circleci.com/product/

work on the exercise in their own time if they prefer. We ask them to work in groups of 4, although if we had a smaller class we might recommend working in pairs. Then during the following week, the work of each group is assessed by a marker during a 20-minute live review session. Students can book a convenient slot for them using an online diary. As of January 2020, with 52 groups and three markers working in parallel to assess roughly the same number of groups each, the whole assessment process took a total of 6 hours (so 18 assessment hours in total, spread over three days). During each live review, the marker and the group members discuss:

- **Continuous Deployment** - *do failing tests successfully block a bad change from being deployed to production?* To answer this, the students are asked to demonstrate that, after introducing a change that forces one of their tests to fail, their pipeline stops at the test stage, and no new deployment occurs until the tests are fixed.
- **Docker and Heroku** - *what are the benefits of deploying a containerised application? And what are those of deploying to a PaaS provider like Heroku?* The students are expected to show an understanding of the advantages of these technologies in terms of portability, maintainability, scalability and service availability.
- **Scripting solutions** - *how is the application deployed to the virtual machine? Is sensitive data (like Heroku's API key) exposed as plain text or is it rather stored in a secret variable?* The configuration files and associated scripts are reviewed, with particular focus on how the Java application is launched during the VM deployment. Ideally, the application is launched in the background, with standard output and error streams redirected to a file. A strategy should also be devised for cleanup of old instances of the application on every new deployment. Some security aspects are also considered with respect to the VM firewall (did the students create a firewall exception for the port their app listens to, or did they simply disable the firewall entirely?), the system privileges of the GitLab Runner user (did the students grant root privileges to this user to run certain tasks?), etc.
- **Extensions** - At the end of the review session, each group is invited to showcase any work completed beyond the exercise's specification. This might be UI and/or testing improvements to the application itself, or any of the extensions to the deployment pipeline discussed earlier in Section III-D. Any extra work falling in either of these broad categories is considered for credit, but given the aims of the exercise, we decided to award more marks for CI/CD-specific extensions.

IV. Later Projects

Having completed the *DevOps Lab* exercise, we expect students to be able to set up a CI/CD workflow for projects related to other courses in their degree. In particular, we explicitly ask students to demonstrate DevOps practices during the two group projects that follow later in Year 2. The first of these projects supports the Compilers course, and involves writing a compiler for a simple programming language from scratch. A compiler obviously takes quite a different form from the simple web application discussed previously, but students are able to transfer their knowledge and skills and set up tests and a pipeline appropriate for the design of a compiler. Later in the year the students undertake a more free-form web development project, in a module where the focus is on agile product development and human-centred design. Students are free to use whatever technology they choose, but we require them to demonstrate their use of continuous delivery practices and regular progress through working software, which they are to demo on a weekly basis in a production environment. Again, students are able to reuse, and build upon, the skills from the *DevOps Lab* to support this.

The following year, as Year 3 students, they will undertake a larger team project, again to be delivered iteratively. Each team works on a different project with one or more clients and adopts a wide variety of tools and platforms as best fits the software they are building. At this point we do not make it an explicit requirement that teams follow and demonstrate DevOps practices, but we observe that almost every team sets up an appropriate infrastructure to help them build, test and release their software, having seen the value of this in their Year 2 projects.

V. Discussion and Conclusions

We are certainly not the first university to introduce the topic of DevOps into its Computer Science curriculum. Several other universities advertise modules on this topic, including Glasgow Caledonian University[12], Cardiff University[13] and Johns Hopkins[14]. These modules often target masters-level students and are typically organised as complete modules, with lectures, tutorials and final exams. This allows for a more in-depth discussion of the theory than with our approach, but in doing so perhaps presents the topic as something "advanced" to be learned in isolation after more fundamental software engineering courses have been completed. In contrast, our approach favours a more hands-on introduction to DevOps practices earlier in the degree followed by continuous application of the practices through subsequent projects, presenting it as something fundamental on top of which other software engineering practices can sit.

Work at the University of Luxemburg [12] seems closer to what we have done, given its practical focus, but it is still structured as a traditional university module that spans several weeks. As described in Section III-E, the *DevOps Lab* lasts a single week (or two, if we consider the code review sessions to be part of the learning process), which we have found to be just the right amount of time for our students to pick up the essentials of DevOps, and build a basic set of skills. After this,

[12]https://www.gcu.ac.uk/study/modules/info/?Module=M3I325687

[13]https://www.cardiff.ac.uk/study/undergraduate/courses/2020/applied-software-engineering-bsc

[14]https://apps.ep.jhu.edu/course-homepages/3562-605.609-devops-software-development-garonzik

students are equipped to develop their skills and knowledge through application in different modules across the curriculum.

Another interesting approach is to teach DevOps practices by means of a dedicated web-based platform, for example ALECSS [13] or DevOpsEnvy [14]. In the case of DevOpsEnvy, the platform acts as a façade for a variety of open source tools, such as Jenkins, Docker and SonarQube , hiding away their configuration complexities. This way of lowering the barrier to entry for first experiments sounds enticing; however, in our case we wanted to expose students directly to the real tools. We wanted to support re-use of the acquired skills in many varied projects, which would likely not be possible if too many of the details were abstracted.

We have consciously decided not to merge the teaching of technical DevOps practices with that of agile development processes and human-centred design principles. We found that students found it hard to learn all of these concepts together in one project, and so the discussion of agile development and HCD is postponed until the summer term of Year 2. By this time the students are confident working with the technical side of writing tests and setting up build and deployment pipelines, leaving them more head space to think about designing a product and managing an agile project.

In our *DevOps Lab* we aim to train the students to read and understand the official documentation and configuration settings of the tools they choose. This has worked well over the past two runs of the *DevOps Lab*: after a few hours (on average) of a moderately steep initial learning curve, the students show a remarkable improvement in their ability to navigate new documentation and get started with new tools, as they can draw from the experience of having already gone through the whole configuration and set-up processes with similar services. Students who took part in the *DevOps Lab* in Spring 2019 and undertook an internship over the following summer reported that the skills they had acquired during the course helped them speed up the technical on-boarding process in their respective companies and fit easily into the ways of working of a professional team.

Overall we are encouraged by the results of using a short practical exercise early on in the degree programme to introduce DevOps skills and techniques, which we believe to be fundamental to engineering quality software. Although tools and technologies will inevitably change with time, we hope that instilling students with these ways of working will serve them well during their future careers as software engineers.

References

[1] M. Kropp, A. Meier, and R. Biddle, "Teaching agile collaboration skills in the classroom," in *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*, April 2016, pp. 118–127.

[2] Z. Masood, R. Hoda, and K. Blincoe, "Adapting agile practices in university contexts," *Journal of Systems and Software*, vol. 144, pp. 501 – 510, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121218301419

[3] J. Campbell, S. Kurkovsky, C. W. Liew, and A. Tafliovich, "Scrum and agile methods in software engineering courses," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, 2016, pp. 319–320.

[4] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010.

[5] N. Forsgren, D. Smith, J. Humble, and J. Frazelle, "2019 Accelerate State of DevOps Report," Google, Tech. Rep., 2019. [Online]. Available: http://cloud.google.com/devops/state-of-devops/

[6] G. Kim, P. Debois, J. Willis, and J. Humble, *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, 2016.

[7] N. Forsgren, J. Humble, and G. Kim, *Accelerate: The Science of Lean Software and DevOps Building and Scaling High Performing Technology Organizations*, 1st ed. IT Revolution Press, 2018.

[8] R. Chatley and T. Field, "Lean Learning - Applying Lean Techniques to Improve Software Engineering Education," in *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)*, May 2017, pp. 117–126.

[9] 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*, ser. XP 16 Workshops. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2962695.2962707

[10] N. Abbas, A. M. Gravell, and G. B. Wills, "Historical roots of agile methods: Where did "agile thinking" come from?" in *Agile Processes in Software Engineering and Extreme Programming*, P. Abrahamsson, R. Baskerville, K. Conboy, B. Fitzgerald, L. Morgan, and X. Wang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 94–103.

[11] G. Kim, K. Behr, and G. Spafford, *The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win*, 1st ed. IT Revolution Press, 2013.

[12] E. Bobrov, A. Bucchiarone, A. Capozucca, N. Guelfi, M. Mazzara, and S. Masyagin, "Teaching DevOps in Academia and Industry: Reflections and Vision," in *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, J.-M. Bruel, M. Mazzara, and B. Meyer, Eds. Cham: Springer International Publishing, 2020, pp. 1–14.

[13] M. Ohtsuki, K. Ohta, and T. Kakeshita, "Software Engineer Education Support System ALECSS Utilizing DevOps Tools," in *Proceedings of the 18th International Conference on Information Integration and Web-Based Applications and Services*, ser. iiWAS 16. New York, NY, USA: Association for Computing Machinery, 2016, p. 209213. [Online]. Available: https://doi.org/10.1145/3011141.3011200

[14] G. Rong, S. Gu, H. Zhang, and D. Shao, "DevOpsEnvy: An Education Support System for DevOps," in *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEET)*, Nov 2017, pp. 37–46.