Project based learning for DevOps: School of Computing experiences

Miloš Radenković

School of Computing
Union University
Belgrade, Serbia
mradenkovic@raf.edu.rs
0000-0002-1708-9799

Snežana Popović

School of Computing
Union University
Belgrade, Serbia
spopovic@raf.edu.rs
0000-0002-6774-670X

Svetlana Mitrović

Faculty of Project and Innovation Management
EDUCONS University
Belgrade, Serbia
svmitrovic@mts.rs

Abstract— The subject of this paper is the implementation of DevOps practices in an academic environment, specifically at the School of Computing. The goal is to offer insights into the implementation process alongside the chief challenges that needed to be overcome. DevOps approach to software development is quickly becoming an industry standard, besides its technological aspects such as tools and pipelines, DevOps also brought changes in culture necessitating tight collaboration from team members. Despite it being an industry standard, most universities are yet to include DevOps practices in their curriculums. For this purpose we propose a model for incorporating DevOps practices in universities based on the advances made at the fourth year of studies at the School of Computing, where students were introduced to some CI/CD aspects during the course of a semester long project. We offer some insights into potential ways to better facilitate student collaboration, and ways to guide them towards best practices already realized in the industry for bridging the gap between the academic and corporate environments.

Keywords— project-based learning, DevOps, software engineering.

I. INTRODUCTION

Business demands for scalability, availability and rapid development of applications is constantly on the rise. To meet these rising business demands, new methoologies and tools are needed. One of the most popular approaches today that seeks to meet these rising demands is DevOps[1]. DevOps is a way of improving the software development process through the concepts of continuous development, integration, testing, delivery and monitoring[2].

Despite the popularity of DevOps in the business environments, the concepts that it relies upon are often overlooked in university education[3]. This problem manifests itself in the ever rising demands for new software developers who are capable of working in DevOps environments, but the universitiy curriculums are falling behind and leaving the gap between academic and business environments ever larger. While there are many attempts to implement devops in education[4][5], there hasn't been a global shift in educational pririties.

In order to overcome these problems new technologies and concepts must be introduced into the educational process[6]. But the knowledge gap is not the only problem with the educatinal process, DevOps environments heavily rely on collaboative culture as a cornerstone of problem solving. To overcome this problem a pedagogical shift must occur, where individual problem solving must be directed towards problems encountered in the idustry and complex problems overcome through teamwork and sophisticated tools.

In this paper we present an approach for introducing DevOps principles in senior years of university education. The approach was evaluated at the School of Computing. In order to better show the required changes to the educational process we will highlight the way the course was organized and the tools which were used to guide students through CI/CD (continious integration and continious delivery) and facilitate the change in student collaborative culture.

II. DEVOPS CONCEPTS FOR SOFTWARE ENGINEERING EDUCATION

The term DevOps stands for combination of development(Dev) and IT operations(Ops). By combining these two segments which were traditionally kept separate allows for higher levels of collaboration and removes some of the most common problems that arise when teams are not aware how the other performs their tasks[7]. This paradigm shift means that teams are becoming multifunctional and are made up of members with different qualifications. While this change in culture is critical its impact would fall flat if it was not assisted by the various tools that better facilitate this collaboration and in optimal cases even automate it[8]. This high level of automation is in fact something that is most associated with DevOps, but without changes in culture that facilitate the use of these automation tools, their effect would be somewhat limited[9].

These automation tools and changes in culture are not the only things that are required for a successful DevOps implementation. In order to fully realize DevOps projects, they need to be based on current technologies that allow

for applications, in the form of services[10][11], to be packaged into containers for ease of testing and deployment[12]. These containers rather than source code are the building blocks of DevOps projects and allow for such high levels of automation and make deployment and scaling of applications a standardized process. Likewise to fully capitalize on the automation aspects and the scalability aspects, it is preferable to use microservice architectures, where each microservice can be built, containerized, tested and deployed in an automated manner.

The usual DevOps lifecycle is made up of eight distinct phases: planning, coding, building, testing, release, deployment and monitoring[13]. In an ideal environment all the activities besides planning and coding can be automated or at least made to work with minimal human guidance. While DevOps is not in the strictest sense a project management methodology, it relies on the project being managed in an agile manner such as SCRUM where rapid iterations where automation can be fully leveraged[14][15].

III. PROJECT-BASED LEARNING BASED ON DEVOPS

The DevOps approach was evaluated at the School of Computing, at the fourth year of studies on the subject of "Software Engineering". The course had approximately 100 students enrolled which were further divided into four teams, of 25 students each. These teams were formed with the idea of tackling a single project during the course of the semester, the performance on this project would play the main role in their grade and participation was mandatory. Since student teams were relatively large in size, detailed coordination by the professors would be difficult, for this reason students were required to self-organize and were provided with considerable autonomy. Projects themselves were open-ended in the sense that the students were provided with a topic such as "Banking", "Insurance", "Hospital", "Accounting". The only restrictions to the projects were technical in nature and mostly dealt with technologies, frameworks and system architectures. Students were encouraged to form smaller teams to fulfill more specialized roles, and most teams settled into the traditional teams of: Backend team, Frontend team, Specification team.

Students were divided into four roles:

Project Manager were tasked with keeping track of the project itself. All the communication with professors was through the managers, necessitating that student teams filter up any necessary information to manager level. Managers were tasked with keeping detailed accounts on the activities of individual team members with the help of team leaders. In keeping with the autonomous nature of teams, managers would also grade their team members, and these grades would be used as the main component in the students grade. Given their ability to directly influence someone's grade, managers were to play the a direct role in the success of the project.

Assistant Managers were necessary due to the overwhelming nature of the role of Project managers, their task was to assist their manager with any required tasks, and were given many powers to do so. The chief distinction between managers and assistant managers was that only the project manager could assign grades and communicate directly with professors. Assistant managers also primarily dealt with scheduling of meetings and other purely organizational tasks.

Team leaders were tasked with running teams of between 5-10 team members. Seeing as most teams dealt strictly with programming, the main role of team leaders was to assign tasks to individual members, and to review their code once the tasks were complete. Team leaders rarely programmed themselves, and were usually more experienced programmers that guided their team members and helped them in their various tasks. Team leaders were also tasked with informing managers on the state of their tasks, and the activities of their team members.

Team members were tasked with completing their tasks within the assigned deadlines, and were encouraged to participate in as many meetings as possible.

A simplified view of the main interactions within the framework can be seen on figure 1.

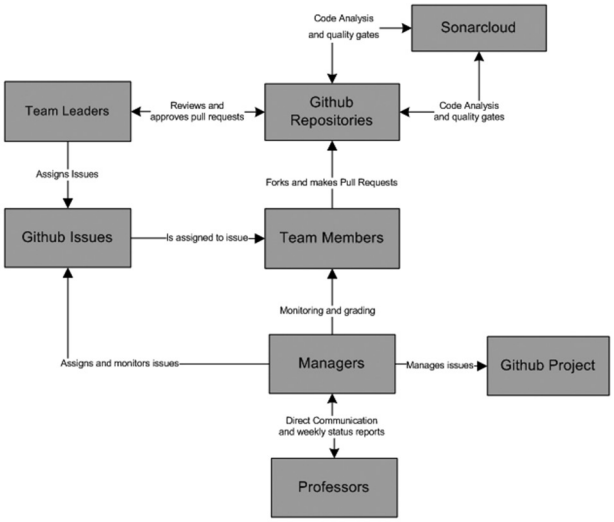

Figure 1. A simplified view of main interactions between roles

Since to goal of such organization was to facilitate autonomous organization and close collaboration within teams, students were free to manage their projects as they saw fit. With the topics provided it was up to the students to research how such systems functioned and to provide their own specifications for them. These gathered specifications were presented to the professors at regular intervals for approval in order to maintain the required complexity of the projects as well as to maintain that the systems were faithfully represented. Students were free to use any project management methodologies that they

saw fit, but all teams settled on agile methods, incorporating most SCRUM good practices in their projects, such as one week sprints and regular short-length meetings. In order to better capitalize on the agile nature of their projects, regular control points of their projects were organized. Each week every team would present their progress to the professors, and any potential issues within the teams would be addressed and questions answered. In addition to the weekly control points, there were deliverable control points where students had to present their code and applications, in order to gain points which are to be allocated to their project as a whole. The points allocated in such a way were at the project managers disposal for further allocation to individual team members based on their effort up to that point. In addition to the usual deliverables that were made for 25%, 50% and 100% of project completion, students were also encouraged to tackle additional tasks (mostly dealing with CI/CD) which would likewise increase their collective points.

In order to teach the students more about DevOps workflows and organization. Teams had many tools at their disposal and were taught and encouraged to incorporate them into their projects as much as possible. Some of these tools include:

Mattermost – was used as the primary means of communication within the project, and only mattermost communication was considered to be “official” by the teaching staff. Channels were formed around individual teams(for example Backend), likewise channels were made for cross team collaboration such as (back/front, back/spec, etc..). Mattermost was also used for all official announcements by the professors, and managers were also encouraged to use mattermost for any official announcements. An example of team channels for one of the project groups can be seen of figure 2.

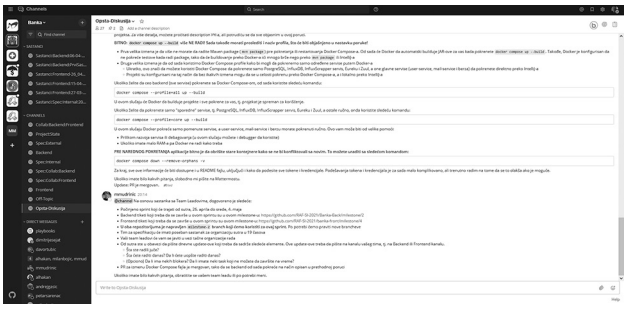

Figure 2. Team chat with its relevant channels

BigBlueButton – is a video conferencing software, and was used to facilitate student meetings. All student meetings were made through BigBlueButton and were saved by the platform for later viewing by either students who could not make it to the meeting, or the professors. Bigbluebutton was further integrated with mattermost, where each bigbluebutton meeting would be assigned its own mattermost channel, where the access link could be found, and the recording to the meeting. By integrating these two tools, we have made sure that all project communication was transparent to all those involved, and hopefully minimized any problems that could arise due to access. An example of integration between BigBlueButton and Mattermost can be seen on figure 3.

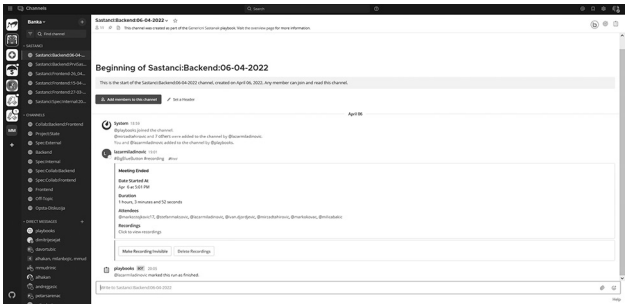

Figure 3. Mattermost-BigBlueButton integration

Google Calendar – In order to ovecome scheduling conflicts that come with such large groups, all students had to keep their google calendars up to date. Google calendars were the primary means of sharing invites for Bigbluebutton meetings and were critical in ensuring that timeslot could be found where everyone was available.

Github – every project had two separate repositories, one for frontend and the other for backend development. Only the project managers and team leaders had to permissions to create, merge and push to branches. All other team members had to use forks and pull requests in order to submit code for potential addition to a branch. A system based on pull reqests meant that team leaders could play the role of code reviewers before accepting any pull requests that were of sufficient quality. Additionally these pull requests were usually made by a single team member, resulting in a highly transparent system where the contributions of individual team members could be clearly measured.

Github project and github issues – students were encouraged to use github issues as project activities that could be assigned to any individual team member. Then as pull requests were made by those team members, they could be tied to those issues. Github project was used as an alternative to conventional project management applications such as openproject. The use of github issues with github project allowed for a very fine grained approach to project tracking. Likewise, being hosted on github allowed these github projects to be viewable by both the professors and other team members, keeping the state of the project public for all. One of the student github projects with its active tickets can be seen on figure 4.

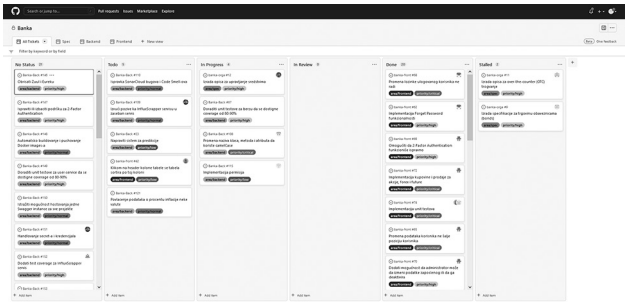

Figure 4. Github project from one of the student teams

Docker – In order to achieve continuous delivery, all microservices within their projects had to be containerized. For development purposes docker and docker-compose tools were used. Docker-compose allowed the students to live test their applications, where whole microservice stack could be rapidly initialized on any machine. Docker-compose was also used for live demonstrations of projects during classes.

Kubernetes – Since docker-compose was used for development environments, docker images were already developed and uploaded to docker repositories such as dockerhub. The existence of these docker images made it easy to integrate them inside a Kubernetes cluster where production environments could be simulated, and continuous delivery workflows integrated. All projects were provided with their own kubernetes cluster made up of several nodes hosted on the private cloud.

Sonarcloud – in order to better facilitate continuous integration, testing and code quality was partially automated by integrating Sonarcloud service with github repositories. This integration allowed each pull request to be analyzed by sonarcloud, to highlight bugs, failed tests, errors, and bad quality code. Sonarcloud reports have proven to be an invaluable tool for team leaders that needed to review and approve pull requests.

IV. ANALYSIS AND CONCLUSION

As of the writing of this paper the projects are still ongoing, but as projects are organized around smaller deliverables it was possible to analyze them as seperate lifecycles in order to improve and further adapt the approach for the coming iterations. Some of the main takeaways from the projects are the following:

Students had limited knowledge on how software should be tested and were averse to writing tests. After additional effort was made to ensure that all team members were educated on writing tests for their own forks they are starting to realize how important testing is to the the project lifecycle. Team leaders have further embraced testing as a way of ensuring that the code they are reviewing is properly tested before it is submitted to them. In order to further reinforce the newly established testing practices a requirement for 80% test coverage for all new microservices was established.

If suitable infrastructure is provided in the form of mattermost and bigbluebutton. Students will utilize this infrastructure to its fullest. Additionally, transparency in the form of recordings of their meetings plays a large factor in their use of the platforms. Likewise github in the form of issues/project can be used for barebones project management purposes, and has proven to be more than up to this task especially when its close integration with repositories is considered. Overall up to now, at the 50% project progress students have made 153 BigBlueButton video conferences with an average duration of 1 hour. A short overview of mattermost activity for one of the teams can be seen on figure 5.

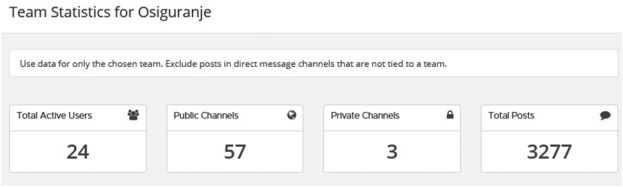

Figure 5. Team statistics for one of the four teams

The project should have many deliverables, which encourages the students to be even more agile in their tasks. In particular the deadline for the first deliverable should be within the first two weeks of the course. This first deliverable should be centered around common task such as user management, the goal of this first deliverable is to ensure that backend and frontend development do not wait too long for the first specificactions from the specifications teams to arrive. By the time they are done with user-management services, they will have established the necessary workflows and knowledge of tools in order to tackle the more difficult tasks in the specifications.

There is a large disparity in both aptitude and knowledge when backend and frontend tasks are concerned. All students were interested in being a part of backend team while interest in frontend development was lacking. When left to choose their own teams, backend team members outnumbered the frontend members at a factor of 2:1. After the first deliberable this problem became evident to the project management, and team „rebalancing“ took place. After the rebalancing took place, the new frontend team members had to be taught by the frontend team in order to become productive team members. This disparity in both aptitude and knowledge should be addressed as soon as possible, so that „training“ can be done in the earliest iterations so as not to endanger the capablities of the frontend teams.

By using docker and containerizing their microservices, students can truly view the rest of the project as a black box into which the microservice they are developing can be plugged in. No matter the size of the application, all developers can run it through docker-compose and run integration tests with other services. For this reason it is important that students start containerizing their services from the start of their projects.

By having such large teams of 25 people collaboration becomes more difficult, but it also reinforces the idea that good collaboration is critical to the success of the project, and student teams will often see this for themselves and will seek to improve their collaboration as time goes by. Having such large teams also means that there is a large disparity in knowledge and experience between students. Since collaboration is critical to the success of the projects the students have taken upon themselves to teach each other the necessary technologies and good practices. This knowledge transfer is further aided by bigbluebutton and the fact that all meetings are recorded and can be accessed

at any time.

Since collaboration is critical to the success of the projects, naturally teams where team members know each other are naturally performing better than teams where there is little prior knowledge among team members. Some teams have recognized this problem and have started the practice of team building in order to improve collaboration and make communication between team members easier.

While the managers have the means of negatively influencing someones grade, this mechanism was often not utilized, and when it was utilized it often had a negative effect on the team morale. For this reason teams have realised that they need to focus on improving their own collaboration in the form of better detection of delays and misunderstandings. This has resulted in iterations becoming shorter and tasks becoming more manegable and less dependant on each other.

Collaboration between frontend and backend teams broke down in periods of high activity in proximity to the deadlines. In order to counteract this problem students have realised the importance of documenenting their services and have started using tools such as swagger. Additionally regular meetings between frontend and backend teams have been established as the norm, where backend teams present the developed services, and frontend details how they would wish their data to be delvered.

By highlighting the most common issues faced by large project teams and the solutions to these problems we hope to provide any future implemenetations of DevOps in education with a suitable framework upon which they can build their own courses. Likewise we presented a set of tools which were made at the students disposal and have proven critical to the success of the project teams. Our experiences at the school of computing at the senior year also show that when presented with suitable tools and the knowledge on how to use them, students were able to quickly adapt developing within CI/CD and working within large teams in order to complete complex projects.

REFERENCES

[1] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, “DevOps,” IEEE Softw., vol. 33, no. 3, pp. 94–100, May 2016.

[2] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, “A Survey of DevOps Concepts and Challenges,” ACM Comput. Surv., vol. 52, no. 6, Nov. 2019.

[3] Z. Bogdanović, M. Despotović-Zrakić, T. Naumović, L. Živojinović, and A. Bjelica, “Inducing creativity in engineering education: A crowdvoting approach,” in 2019 18th International Symposium INFOTEH-JAHORINA, INFOTEH 2019 - Proceedings, 2019, no. March, pp. 20–22.

[4] H. B. Christensen, “Teaching devops and cloud computing using a cognitive apprenticeship and story- Telling approach,” Annu. Conf. Innov. Technol. Comput. Sci. Educ. ITiCSE, vol. 11-13-NaN-2016, pp. 174–179, Jul. 2016.

[5] R. A. K. Jennings and G. Gannod, “DevOps - Preparing Students for Professional Practice,” Proc. - Front. Educ. Conf. FIE, vol. 2019–October, Oct. 2019.

[6] M. Fernandes, S. Ferino, U. Kulesza, and E. Aranha, “Challenges and Recommendations in DevOps Education: A Systematic Literature Review,” ACM Int. Conf. Proceeding Ser., pp. 648–657, Oct. 2020.

[7] T. Cardoso, R. Chanin, A. Santos, and A. Sales, “Combining Agile and DevOps to Improve Students’ Tech and Non-tech Skills,” vol. 1, no. Csedu, pp. 299–306, 2021.

[8] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, “Software Engineering for DevOps,” Ieee Comput. Soc., no. June, pp. 94–100, 2016.

[9] M. Kersten, “A cambrian explosion of DevOps tools,” IEEE Softw., vol. 35, no. 2, pp. 14–17, Mar. 2018.

[10] L. Chen, “Microservices: Architecting for Continuous Delivery and DevOps,” Proc. - 2018 IEEE 15th Int. Conf. Softw. Archit. ICSA 2018, pp. 39–46, Jul. 2018.

[11] A. Balalaie, A. Heydarnoori, and P. Jamshidi, “Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture,” IEEE Softw., vol. 33, no. 3, pp. 42–52, May 2016.

[12] H. Kang, M. Le, and S. Tao, “Container and microservice driven design for cloud infrastructure DevOps,” Proc. - 2016 IEEE Int. Conf. Cloud Eng. IC2E 2016 Co-located with 1st IEEE Int. Conf. Internet-of-Things Des. Implementation, IoTDI 2016, pp. 202–211, Jun. 2016.

[13] M. Virmani, “Understanding DevOps & bridging the gap from continuous integration to continuous delivery,” 5th Int. Conf. Innov. Comput. Technol. INTECH 2015, pp. 78–82, Jul. 2015.

[14] L. E. Lwakatare, P. Kuvaja, and M. Oivo, “Relationship of DevOps to Agile, Lean and Continuous Deployment,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 10027 LNCS, pp. 399–415, 2016.

[15] A. Hemon, B. Lyonnet, F. Rowe, and B. Fitzgerald, “From Agile to DevOps: Smart Skills and Collaborations,” Inf. Syst. Front., vol. 22, no. 4, pp. 927–945, Aug. 2020.