# Training Software Engineers using Open-Source Software: The Professors' Perspective

Gustavo Pinto
Federal University of Pará
Belém, PA, Brazil
gpinto@ufpa.br

Fernando Figueira Filho
Federal University of Rio Grande do Norte
Natal, RN, Brazil
fernando@dimap.ufrn.br

Igor Steinmacher
[1]Northern Arizona University
Flagstaff, AZ, USA
[2]Federal University of Technology
Campo Mourao, PR, Brazil
igorfs@utfpr.edu.br

Marco A. Gerosa
Northern Arizona University
Flagstaff, AZ, USA
marco.gerosa@nau.edu

***Abstract*—Traditional software engineering courses often prioritize methodologies and concepts in small, controlled environments: naive projects used as a proof of concept instead of full-fledged real software systems. Although this strategy has clear benefits, it does not place enough care in training students to face complex, non-trivial legacy software projects. To bridge this gap, novel software engineering courses are leveraging the rich variety of open-source software (OSS) to illustrate how these methodologies and concepts are applied into existing, non-trivial software systems. To better understand the benefits, challenges, and opportunities of this transition, in this paper, we interview seven software engineering professors that changed their academic setting to aspire students to comprehend, maintain, and evolve OSS systems as part of their software engineering course. We found that there are different ways to make use of OSS projects in SE courses in terms of project choice, assessment, and learning goals. Moreover, we evidence clear benefits of this approach, including improving students' social and technical skills and helping students enhancing their resume. Also, we observed that this strategy comes with costs: the activity demands effort and time from the professor and the barrier for one getting involved with and, therefore, placing a meaningful contribution, in an OSS community is often high.**

***Index Terms*—Open-source software; Teaching Software Engineering; Open-Source Contributions;**

## I. Introduction

The steady flow of new techniques, tools, and processes for developing software systems along with the interdisciplinary nature of software development, makes the software engineering discipline one of the most challenging to teach and learn [1], [2]. On the one hand, there is the traditional software engineering courses, which emphasize the methodologies and theoretical concepts, but pay little attention on teaching students how to deal with complex, existing software systems [3]. On the other hand, there is the software industry, which is eager to hire software developers that can cope with legacy code, but has little time available for training unskilled yet promising software developers.

In order to bridge this chasm, one approach that is gaining more supporters lately is to bring to the classroom real, non-trivial problems that the software industry face, and stimulate students to contribute to them [4]–[8]. Since it is not always straightforward to bring proprietary projects to the classroom, *e.g.*, they often rely on restrictive software licenses, professors are taking advantage of Open-Source Software (OSS). The motivation for using OSS is three-fold:

1) There is a plethora of open-source software choices, with different domains, sizes, and complexities;
2) Well-known OSS projects often exhibit the maturity and the breadth of scope necessary to use in real-world problems;
3) Well-known OSS are maintained by an active global community of software developers.

Therefore, exposing students to OSS might give them the opportunity to face and, eventually, overcome social and technical barriers that new contributors face while joining software projects [9], [10].

This approach leads students and professors to significant learning experiences. From the students' perspective, this approach is beneficial since working on OSS projects enables them to learn real-world skills, attitudes, and experiences [4], [11], which might increase their confidence, for instance, when applying for industry jobs. From the professors' perspective, it helps them to achieve the goal of preparing the future workforce of software developers, for instance, since it is possible to bring to classes real world practices, problems, and solutions. However, this approach can also hide some drawbacks [12], [13]. For instance, students might face barriers for (1) getting involved with OSS communities and/or (2) finding and placing a meaningful contribution during the often short term of a software engineering course. These facts might demotivate students. Similarly, professors need to deal with things that are often beyond of the scope of an ordinary class, such as unfriendly open-source communities or an in-depth knowledge of different open-source softwares under study.

To understand the hidden benefits and challenges of such practice, in this paper, we interviewed seven software engineering professors that recently changed their courses to foster open-source initiatives. Since this research is still in its early stages, in this paper we focus on three high-level research questions, regarding the choice of a suitable open-source project to students work on, and the benefits and limitations of this practice. Specifically, we propose to answer three research questions (RQ):

- **RQ1.** What makes a good open-source project for training SE students?
- **RQ2.** From the professors' perspective, what are the benefits of exposing students to OSS development?
- **RQ3.** From the professors' perspective, what are the challenges associated with exposing students to OSS systems?

To provide answers to these questions, we followed a qualitative research method. The insights obtained from this research provide important lessons learned for professors, students, and OSS communities that want to take advantage from this process. The main findings of the paper are the following:

- We found that it is important to give enough freedom to students choose an open-source project to work on, although restrictions such as project popularity and relevance should also be taken into account.
- We observed that contributing to open-source improves not only technical skills. Successful students that overcome the first contribution barrier also developed their social skills.
- We noted that the time constrains of a course can not only delay the contribution process, but also make it difficult to students engage with open-source projects.

The rest of the paper is organized as follows: Section II discusses related studies and how this study differs from them. Section III describes the methodology employed, and Section IV provide the main findings of this research. Further, Section V envisions some implications of this study, while acknowledging some limitations. Finally, Section VI concludes this work and sheds some light for future work.

## II. Related Work

Several recent efforts have involved students in OSS projects within the classroom [4]–[6], [8]. Smith *et al.* [5] focus on selecting the most effective OSS systems to students work with. The authors argued that, due to the short duration of a typical software engineering course (4-5 months), SE professors should not select large or complex OSS projects, because students would have a hard time in placing a meaningful contribution during the course.

Similarly, related work suggests that OSS projects should not be too small or too naive, since students would not take advantage to use SE principles. Morgan and Jensen [4] detailed the experience of teaching software engineering courses with the support of OSS. The authors also observed that the size of the OSS project (in that case, Ubuntu) turned out to be a significant obstacle for students placing a contribution (*e.g.,* unable to find a simple yet interesting task to work on). Likewise, Buchta and colleagues [6] reported other experiences of teaching the evolution of OSS in software engineering courses. The authors reported a setup cost (*e.g.,* selecting OSS projects, setting up the version control system) of about 60 hours before the course take place. In this study, however, the authors did not discuss the benefits, implications, or challenges of using this methodology for running a software engineering course. Moreover, Holmes and colleagues [14] reported that in their Undergraduate Capstone Open-Source Projects (UCOSP) program, open-source projects under study must maintain an active issue tracker, use a version control system, and perform code reviews. They also reported that each project should have at least one mentor who will help students throughout the course. In the study of Petrenko *et al.* [7], the authors described their grading system: students are graded based on their individual effort, which include only implementing, debugging and testing changes, but also justifying their architectural decisions. In the work of van Deursen *et al.* [15], grading is based on team (*e.g.,* presentations, code analysis, etc) and individual deliverables (*e.g.,* personal review, participation in lectures, etc).

Although open-source software has been a formal presence in software engineering education during the last decade [16], [17], most of the work published in this area were experience reports (*e.g.,* [4]–[6], [14]). Therefore, these reports do not capture the whole spectrum of dynamics that may arise in these courses. Also, since these studies ask different research questions, the findings are not unified and are hardly compared. Still, the recent introduction of social coding websites such as Github and Bitbucket changed the way software engineers build software (and, consequently, how professors use OSS in their courses). Therefore, studies that focus on the traditional patch-based model (*e.g.,* [7]), might not be generalized to the pull-request based era [18]. In fact, the professors' perspective that employ these platforms are particularly relevant to our study.

The studies presented in this section are experience reports based on the own professor's perspective, which is rather limited. To the best of our knowledge, our work is unique with the focus on bringing different professors (with their settings, perspectives, etc), to answer the same set of research questions. We believe this approach has the potential of providing a broad and deep understanding of this emerging field.

## III. Methodology

To provide answers to the research questions stated in Section I, we conducted interviews with software engineering professors. The interview procedure, as well as the qualitative coding approach are detailed in this section.

### A. Interviews with Professors

We conducted semi-structured interviews with professors to understand how assignments using OSS projects are used in software engineering courses. We used a snowballing approach to recruit professors that empy such approach. We identified and interviewed a total of 7 faculties. The interviews were conducted by phone. Interviews lasted approximately 40 minutes and audio was recorded. The professors are based on Brazil (4), US (1), The Netherlands (1), and Spain (1). Among them, 5 are full-professors, and the remaining two ones are associate professors. The professors had experiences with OSS projects

in different undergraduate and graduate Software Engineering related courses, including Object-Oriented Programming, Software Architecture, and OSS Development courses. One of the interviewees had an experience with a masters program on OSS. The size of the classes vary from 20 students to 100+ students. There are no off campus students. As the interviewees reported, the majority of their students may not have contributed to open-source before.

The interviews were grounded on RQ1—3. The interviews consisted of three parts: We started asking about the interviewee's background in teaching software engineering and, more particularly, teaching software engineering with OSS. Then, we moved to specific questions, including questions about how professors choose a project, how they create, evaluate, and grade a contribution that students make to an OSS project. In this group of questions, we also asked about the barriers for placing a contribution, motivation, and interests in the course. Finally, we asked questions about the problems the professors' faced with this kind of assignment, what problems students' face, and what are the main advantages and disadvantages.

### B. Interview Analysis

To analyze the data, we first transcribed all the audio files. Each transcript, along with the associated recording, was analyzed by three authors. We then coded the answers and organized them into categories. We followed the guidelines on the open coding procedure [19]. To avoid bias, the coding procedure was done independently, followed by conflict resolution meetings.

## IV. Results

In this section we present the findings of this study, grouped by our research questions. We highlight the main themes that emerged along with quotes from the interviews. Among similar opinions, we chose to quote only the one we considered the most representative for each case. The professors are identified as P1 — P7.

### A. RQ1. What makes a good OSS project for training SE students?

One of the first steps in project work is selecting one or more OSS projects to work with. We asked our interview participants about what makes a good OSS project to be selected and the strategies they use for that. Typically, project selection is somehow mediated by the teacher, who is able to provide guidance in terms of what project is best for the students. However, all of our interviewees declared to provide some degree of freedom for students to choose what project they would like to work with: *"there was a student team that liked networks and they chose NS3, which is a network simulator that has never occurred to me as a project to be selected." [P2]*

Giving students **enough freedom to choose** when selecting an OSS project was regarded as important by all of our interviewees, although typically some criteria need to be met before a decision is reached. For instance, P1 provides their students with a couple of hundred active projects available on GitHub. Once the students pick one project, both professor and students *"briefly go through some recent pull requests, and then get an intuition whether this is sufficiently open and say 'yes you can use this'. [...] It should not be a dead project, with very few contributors." [P1]*

Another criteria for selecting OSS projects were mentioned by P3. He emphasizes the importance of a balance between **project popularity** and **relevance of a contribution**: *"I know that is tempting to choose a popular project, but these tend to be harder to make contributions or to get contributions accepted. If you pick a small project, maybe the contributions would have little relevance." [P3]*

During the selection, sometimes the teacher may act as a proxy by contacting project maintainers directly. For example, P5 does that before letting students select a project. In this context, **fast feedback** from maintainers is crucial since most courses would last only for a semester or a term: *"I say [to the maintainers] that I will contribute to this project with student teams [...] the only thing I ask is that you provide faster feedback than usual." [P5]*

Similarly, before letting students select a project to work with, P4 looks for a **mentor**, *i.e.,* some contact among the project contributors who will help and provide some **recommendations to students**: *"we look for a mentor in the projects. So, we have some recommendations of projects and that made life easier. [...] That's why I wouldn't say it is free to choose because we have our contacts, so we present the projects that are closer to the contacts. "*

### B. RQ2. From the teachers' perspective, what are the benefits of exposing students to OSS development?

Interviewees see this approach as a straightforward way to **engage students with real projects**, "*it is a real professional environment, [...] that's the main [benefit]*" (P4). According to the professors, this is an important advantage of engaging students in OSS projects, partially because of the **level of challenge** imposed by such kind of project. Students need to "*understand what the others had done [...] how is the architecture. This enforces them to develop a much more mature set of skills and reasoning*" (P7). Therefore, by working in such environment, students are exposed to a landscape in which they are required to **develop their technical skills**.

Regarding technical skills, we found that using OSS is a **good way to introduce technologies and tools** that are being used in practice at the moment. One of the professors mentioned that "*it is a good way to teach versioning control systems*" (P5). In addition, it is a good opportunity to have practical experience on different types of frameworks, APIs, or tools. The interviewees also reported that it is a good opportunity to **learn and develop social skills**. Although some contributors are more active than others, OSS projects are usually surrounded and maintained by a community of developers [20]. In order to succeed, the students need to interact with the community, which can occur in multiple ways (*e.g.,* asking questions or explaining their changes). This is

beneficial, according to the professors, because they "*learn how to be part of a team, learn how to be part of a distributed project. They will improve their communication skills*" (P6).

As mentioned earlier, one of the goals of SE courses is to prepare the workforce for software projects. Therefore, interviewed professors made it clear that using OSS is a way to train these students to real environment. This is highlighted by P4: "*[...] one of the good things is that you teach your students how real world software is developed...If you achieve that, then probably is somebody who is prepared to work on a real software development project.*"

Professors also perceived that exposing students to OSS projects is an opportunity to **motivate students**, which was summarized by P6 "*It always feel good to contribute,*" and replicated by P1 "*you can be proud of actually contribute to something*." In this case, students can see it as a starting point, and eventually engage in other OSS communities, benefiting the OSS communities, which might take advantage of additional contributors.

Because of the involvement with a real project, and the open nature of such environments, interviewees reported that it is also **good for the resume**. Students with little or no experience can take advantage of such courses and start creating a portfolio that can be used when applying for industry jobs. P6 highlighted this: "*When they go out and interview with companies, they can say: 'this is the projects I have contributed to. I was part of the team'... You can say: 'look at this code and see what a good developer I am'.*" P5 additionally highlighted that at least one of his students received an offer for the contributions that she did during the course.

We also found that this approach can benefit the students by **supporting the professors** on changing or improving their classes. P2 mentioned that "*new generations cannot sit on a chair for two hours and keep paying attention to a lecture. They lost this ability*". He complemented, saying that "*It is important for the professor to have contact with the real world... The class will be more interesting and useful if he has that experience from outside the university.*"

### C. RQ3. From the teachers' perspective, what are the challenges of using OSS projects?

As mentioned in RQ1, **choosing the right projects** can be a problem, as reported by P7:"*The big challenge is finding projects, or project candidates*," and already indicated in the literature (*e.g.,* [4], [5]). This issue was recurrently reported by the interviewees. To alleviate this situation, P3 mentioned that he cooperates with specific medium port projects that can provide support to the students. With help from internal developers, he identifies tasks that can be performed by the students in one semester. In this case, all students from the class contribute to the same project. Other professors guide the students to choose projects that attend to specific requirement. For example, P1 said: "*the project should be an active project, [...] have multiple pull requests accepted/proposed per week, [...] it should be a large project with several external contributors*." In some cases, professors also provide a non-exhaustive list of recommended projects. Usually, the list provided is based on contacts made a priori with contributors, as reported by P2: "*I usually make contact with key people of some projects making them aware that I will teach this course and asking if they can be student's point of contact*".

The interviewees is the set of **barriers faced by the students** during the contribution process. P4 highlighted that "*[OSS development] isn't a fairy tale*", since it is not always straightforward to have one contribution accepted [21]. The professors acknowledged that the contribution barrier is even more challenging in the context of a SE course. For example, P1 reported that "*[Students'] Technical level could be main problem. Also finding a suitable thing to do is trick*", and P6 said that "*[...] finding a project, understanding the code, finding what work right in the code, submitting and accepting it accepted, putting in the main line of the project, those are the barriers*."

Many interviewees indicated that, depending on the project and the assignment, **time is an issue**. Open-source communities are often overloaded with work, which has the potential of delaying the contribution process. Professors therefore need to realize "**how to make students engage quickly**" (P3). Thus, guidance from the professors is crucial, since "*you may not be able to throw them in the deep end of the pool and say swim*" (P6). However, **professors' time availability** is a also challenge, since it is difficult to follow all students close enough and help them overcoming both technical and social barriers. As mentioned by P3 "*I would have to follow the students closely to try to help them. I noticed that it is necessary, however I had no availability to follow them as close as I wanted to*".

## V. Implications & Limitations

### A. Implications

This research has implications for different kinds of stakeholders. First, SE professors of traditional courses can learn from the experience of their peers. Due to the several advantages found in this new academic setting, such professors can become motivated to bring OSS projects inside the classroom. Still, they can prepare themselves to deal with drawbacks associated with this methodology, for instance, getting in touch with the OSS community before students, and evaluate whether they are friendly or not. Second, since professors have a hard time choosing appropriate projects, students could suggest what kind of projects they are more willing to collaborate (*e.g.,* a project they like most or a project that use a programming language that they are more familiar with). This, in turn, could increase their chances to succeed with the task. Third, OSS communities can also benefit from this study. Since now they know that students are working on OSS projects as part of a software engineering course, they can help them by creating issues that are simple enough to get started yet interesting enough to motivate students. Open-source communities can also get in touch with professors highlighting projects that

are in need of additional contributors. Yet, several students reported problems with getting involved with the community, therefore such communities can leverage open communication channels (*e.g.,* IRC) to foster communication among users. Finally, students can propose this kind of course to professors that are not aware of their benefits.

### B. Limitations

In a study such as this, there are always many limitations and threats to validity. First, we interviewed only seven professors. This happens because some professors have a tight schedule, and were not available for the interview. These professors were found using convenience sampling, which has the chance of limiting the generabillity of our subjects (*e.g.,* $\sim$60% of our interviews are Brazilian professors). Second, and as a result of our first limitation, we did not achieve saturation. That is, it is likely that more themes would emerge if more interviews were conducted. Since this is a work in progress, we plan to further expand the scope and the number of the interviews. Third, the process of deriving themes from the interviews is intrinsically qualitative, thus, human-prone. To mitigate this limitation, the process was conducted in pairs, using well-known qualitative research methods, followed by conflict resolution meetings.

## VI. Conclusions and Future Work

In this study, we interviewed seven software engineering professors that opted for introducing the process of contributing to an existing, non-trivial OSS project as part of a software engineering course. We observed that this process has clear benefits, such as (1) enhancing students' technical skills, (2) enforcing students to learn social skills, and, as a result, (3) improving students' resume. Notwithstanding, this process also hidden some challenges, for instance, (1) choosing the right project to contribute, (2) getting involved with an OSS community, and, as a consequence, (3) finding a representative task to solve during the short term of the course.

For future work, we plan to expand the scope of this study in two ways: First, we plan to conduct additional interviews as a way to refute or validate our additional findings. Second, since students are important stakeholders of the courses, we also plan to get in touch with students that participated in such classes and ask their perceptions about this shift.

## Acknowledgements

We thank the anonymous reviewers for their valuable comments. This work is partially supported by CNPq (406308/2016-0) and PROPESP/UFPA.

## References

[1] Y. Sedelmaier and D. Landes, "A research agenda for identifying and developing required competencies in software engineering." *International Journal of Engineering Pedagogy*, vol. 3, no. 2, 2013.

[2] F. Fagerholm and M. Pagels, *Examining the Structure of Lean and Agile Values among Software Developers*. Cham: Springer International Publishing, 2014, pp. 218–233. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-06862-6_15

[3] O. Cawley, S. Weibelzahl, I. Richardson, and Y. Delaney, *Incorporating a self-directed learning pedagogy in the Computing Classroom: Problem-Based Learning as a means to improving Software Engineering learning outcomes*. IGI Global, 2014, pp. 348–371.

[4] B. Morgan and C. Jensen, "Lessons learned from teaching open source software development," in *10th International Conference on Open Source Systems, OSS 2014, San José, Costa Rica, May 6-9, 2014.*, Berlin, Heidelberg, 2014, pp. 133–142.

[5] T. M. Smith, R. McCartney, S. S. Gokhale, and L. C. Kaczmarczyk, "Selecting open source software projects to teach software engineering," in *45th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '14, 2014, pp. 397–402.

[6] J. Buchta, M. Petrenko, D. Poshyvanyk, and V. Rajlich, "Teaching evolution of open-source projects in software engineering courses," in *22nd IEEE International Conference on Software Maintenance*, ser. ICSM '06, 2006, pp. 136–144.

[7] M. Petrenko, D. Poshyvanyk, V. Rajlich, and J. Buchta, "Teaching software evolution in open source," *Computer*, vol. 40, no. 11, pp. 25–31, Nov. 2007.

[8] D. Coppit and J. M. Haddox-Schatz, "Large team projects in software engineering courses," in *36th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '05, 2005, pp. 137–141.

[9] I. Steinmacher, T. U. Conte, C. Treude, and M. A. Gerosa, "Overcoming open source project entry barriers with a portal for newcomers," in *38th International Conference on Software Engineering*, ser. ICSE '16, 2016, pp. 273–284.

[10] B. Dagenais, H. Ossher, R. K. E. Bellamy, M. P. Robillard, and J. P. de Vries, "Moving into a new software project landscape," in *32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10, 2010, pp. 275–284.

[11] D. M. Nascimento, K. Cox, T. Almeida, W. Sampaio, R. A. Bittencourt, R. Souza, and C. Chavez, "Using open source projects in software engineering education: A systematic mapping study," in *2013 IEEE Frontiers in Education Conference (FIE)*, Oct 2013, pp. 1837–1843.

[12] H. J. C. Ellis, M. Chua, G. W. Hislop, M. Purcell, and S. Dziallas, "Towards a model of faculty development for foss in education," in *2013 26th International Conference on Software Engineering Education and Training (CSEE&T)*, May 2013, pp. 269–273.

[13] H. J. C. Ellis, G. W. Hislop, and M. Purcell, "Project selection for student involvement in humanitarian FOSS," in *26th International Conference on Software Engineering Education and Training, CSEE&T 2013, San Francisco, CA, USA, May 19-21, 2013*, 2013, pp. 359–361.

[14] R. Holmes, M. Craig, K. Reid, and E. Stroulia, "Lessons learned managing distributed software engineering courses," in *Companion Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE Companion 2014, 2014, pp. 321–324.

[15] A. Van Deursen, M. Aniche, J. Aué, R. Slag, M. De Jong, A. Nederlof, and E. Bouwers, "A collaborative approach to teaching software architecture," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '17, 2017, pp. 591–596.

[16] G. W. Hislop, H. J. Ellis, A. B. Tucker, and S. Dexter, "Using open source software to engage students in computer science education," in *40th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '09, 2009, pp. 134–135.

[17] J. Bishop, C. Jensen, W. Scacchi, and A. Smith, "How to use open source software in education," in *47th ACM Technical Symposium on Computing Science Education*, ser. SIGCSE '16, 2016, pp. 321–322.

[18] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE, 2014, pp. 345–355.

[19] R. Hoda, J. Noble, and S. Marshall, "Developing a grounded theory to explain the practices of self-organizing agile teams," *Empirical Softw. Engg.*, vol. 17, no. 6, pp. 609–639, Dec. 2012.

[20] G. Pinto, I. Steinmacher, and M. A. Gerosa, "More common than you think: An in-depth study of casual contributors," in *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016*, 2016, pp. 112–123.

[21] L. F. Dias, I. Steinmacher, G. Pinto, D. A. da Costa, and M. Gerosa, "How does the shift to github impact project collaboration?" in *32th IEEE International Conference on Software Maintenance and Evolution, Raleigh, NC, EUA*, 2016.