Training Software Engineers Using Open-Source Software: The Students' Perspective

Conference Paper · May 2019

DOI: 10.1109/ICSE-SEET.2019.00024

CITATIONS
7

READS
207

5 authors, including:

Gustavo Pinto
Federal University of Pará
78 PUBLICATIONS **729** CITATIONS
SEE PROFILE

Clarice Ferreira
Federal University of Pará
2 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Igor Steinmacher
Federal University of Technology - Paraná/Brazil (UTFPR)
103 PUBLICATIONS **956** CITATIONS
SEE PROFILE

Paulo Meirelles
Universidade Federal de São Paulo
40 PUBLICATIONS **219** CITATIONS
SEE PROFILE

Training Software Engineers using Open-Source Software: The Students' Perspective

Gustavo Pinto[1], Clarice Ferreira[1], Cleice Souza[2], Igor Steinmacher[3], Paulo Meirelles[4]
[1]Federal University of Pará (UFPA), [2]Federal Institute of Pará (UFPA), [3]Northern Arizona University (NAU),
[4]Federal University of São Paulo (UNIFESP)

***Abstract*—Software Engineering courses often emphasize teaching methodologies and concepts in small and controlled environments over teaching, say, maintenance aspects of full-fledged real software systems. This decision is partly justified due to the difficulty of bringing to the context of a classroom a real software project. The widespread presence of open source projects, however, is contributing to alleviating this problem. Several instructors have already adopted contributions to open source projects as part of their evaluation process, and these instructors reported many benefits, including the improvement on students' technical and social skills. However, little is known about the students' perceptions regarding the need to contribute to an open source project as part of a Software Engineering course. To better understand the students' challenges, benefits, and attitudes, we conducted 21 semi-structured interviews with students who took these courses in five different Brazilian universities. We also enriched this data with an analysis of commits performed in the repositories that students contributed to. We observed that even though some instructors chose the open source projects to students to work themselves, some students and even the open source community participated in the process of choosing projects and tasks. Students' contributions varied concerning both complexity (measured by the number of additions, deletions, and edited files) and diversity (measured regarding the different programming languages used). Among the benefits, students reported improving their technical skills and their self-confidence. Finally, some students found extremely important for instructors' being involved with open source initiatives (extra-classroom).**

***Index Terms*—Open Source Software, Open Source Communities, Software Engineering Courses**

I. Introduction

The Software Engineering discipline is one of the most difficult to teach and learn [1], [2], which is mainly due to the constant flow of new technologies and tools that one should master. Traditional software engineering courses often emphasize theoretical methodologies and concepts, over teaching students how to deal with existing and complex software systems [3], [4]. However, although the software industry depends on software developers who know how to handle legacy code, the same software industry has little time available to train unskilled yet promising software developers.

To better align with the software engineering practice, software engineering courses ought to include skills and attitudes that go beyond concepts, methods, and techniques to align with the current software development landscape [5]. This change requires going beyond the boundaries of the traditional way of teaching software engineering, paying more attention to the complexity of social interactions, in particular, discussing how collaborative software development occurs in a real-world environment [6].

However, it is far from trivial to bring software projects developed inside a software company to the context of a classroom due to license issues or the possibility to disclose sensitive information [7]; even to interact with software companies regarding educational matters is not a straightforward task. Thus, a commonly adopted strategy is to conduct collaborative projects, with students working in teams — in some cases in distributed settings, with teams composed of students from different universities. However, these projects are in general *toy projects* tailored to the context of the course; they hardly exhibit maturity or the breadth of scope necessary for use in real software development.

As an attempt to bridge this gap, one approach that is gaining increasing interest is to foster students to participate in Open Source Software (OSS) projects as part of the course [5], [6]. OSS projects are environments inherently collaborative, which stand on the shoulders of a community that interacts to build a software system. Participation in this kind of project enables students to interact with real systems, real problems, and real software development teams interested in building high-quality working software. Thus, students have a unique opportunity to learn attitudes only present in real-world scenarios, which can increase not only their skills but also their self-confidence [8].

In a previous work, we investigated the instructors' perspective on the use of OSS projects in Software Engineering courses [6]. According to the instructors, the benefits of this practice outweigh the challenges given that it improves not only students' technical skills but also develops their social skills. Also, instructors emphasized that the chance of having accepted contributions to recognized OSS projects can be beneficial to compose a portfolio to the students use it in future employment opportunities. Concerning the challenges, instructors reported that the time constraint of a course that can hinder students' engagement with the OSS project. Still, since students are working with particular tasks in many different OSS projects, it is hard for instructors to guide students properly.

As a follow up to our previous work, we sought to investigate students' perceptions regarding this kind of practice. To achieve this goal, we conducted 21 interviews with students who had the opportunity to participate in courses (either in undergraduate or graduate courses) that leverage OSS projects.

The evidence obtained from a qualitative analysis of the interviews yield important lessons that can inspire instructors who want to foster student participation in OSS initiatives. The results may also be of interest to OSS communities that want to take advantage of the students' workforce.

Our main findings include:

1) Choosing the right project and the right task to be accomplished in the course requires a good understanding of OSS practices (e.g., bug trackers, git, or command line interface). However, a task can be better chosen if in collaboration with instructors, students and the OSS community itself (but it would require instructors to be involved with OSS communities);
2) The majority of the students succeed to contribute to OSS. The students' contributions varied regarding complexity (measured by the number of additions, deletions, and edited files) as well as diversity (measured by the domain of the contributed projects as well as the programming languages used). Non-traditional programming languages, such as Scala, Objective-C, or even Elixir, were also explored to perform contributions;
3) The main benefits perceived were associated with the fact of dealing with a real software project and understanding the process of developing real software. However, some challenges still reside such as dealing with complex codebase or the lack of time to contribute;
4) Even though instructors are not experts in the OSS projects, students perceived that the instructors' involvement in extra-class OSS activities (e.g., either doing research or contributing to OSS project) is extremely important to flow the course better.

II. Method

In this section, we state the research questions and the process followed to conduct the interviews.

A. Research Questions

We defined five research questions that guided our study. They are:

RQ1. What are the strategies used to choose the open source projects that students would contribute to?
RQ2. How do the students choose the task they work with?
RQ3. What is the nature of the contributions that students performed?
RQ4. What are the benefits of being exposed to open source software development?
RQ5. What are the challenges of being exposed to open source software development?

The research questions were primarily answered based on the interviews conducted. In particular, we also enriched the RQ3 answer with an investigation of the contributions performed at the studied OSS projects.

B. Interviews

In this section, we describe the procedure followed to select participants for the interviews, how do the interviews were conducted, and how did we analyze the interviews.

1) Participants Selection: We selected the participants based on our previous study [6], which studied the instructors' perspective that leverage OSS projects to students work. Based on the contacts previously established with the instructors, we asked them to contact the students who participated in their courses. By having the contact of those interested, we invited the students to participate in our interview via e-mail. We did that in a way that the number of participants from different universities was similar.

During five weeks, we sent 52 invitations (approximately ten invitations per week). We ended up confirming 20 students interested in participating in our research. Unfortunately, although some students showed *a priori* interest, some of them were unable to participate due to time constraints. We interviewed 16 students. The first two interviews were conducted in a pilot format, aiming to assess the quality and length of the interview script. After these two pilot-interviews, we revised the interview script (we removed some questions as well as inserted and improved others). The two students that participated in the pilot-interviews were discarded from the final set of participants studied. After this first set of interviews were transcribed and analyzed; we conducted seven more interviews. These additional interviews were helpful to substantiate the main categories further already observed, but we were unable to add any new category to our analysis, showing signs of saturation. In the end, we interviewed 23 students (two discarded). Demographics information regarding the 21 participants interviewed are depicted at Table I.

2) Interview Process: We conducted semi-structured interviews using an audio conference software. We recorded the interviews with the participants' consent. We conducted and analyzed the first set of interviews during October and December 2017. The remaining seven interviews were conducted and analyzed between May and July 2018. The 21 interviews lasted on average 49 minutes (min: 24 minutes; max: 77 minutes). The interview script was composed of five parts:

1) In the first part, we asked our interviewees questions about their profile, in particular, which university/course do they had the course, and some initial questions regarding the course (e.g., the period, requirements (if any), and other information);
2) In the second part, we asked the interviewees about the goal of the course, the duration, and which instructor taught the course. We also asked about the student participation in the course and regarding the evaluation format;
3) In the third part, we asked about how do the students chose the OSS project. We also asked about the complexity of the project and the used programming language, as well as, the students' knowledge in that particular programming language. Moreover, how do the task was chosen, which kind of contribution the students made to the selected projects, the interaction with the OSS community, and the difficulties to interact with the communities (if any);
4) In the fourth part, we asked about the contribution pro-

cess and the collaboration with the OSS communities. In particular, we asked our interviewees whether they had already contributed to OSS before the course. We also asked whether they succeed in contributing to an OSS project during the course, the number of contributions made, as well as, whether they had support from OSS maintainers and what were the challenges to contribute to OSS projects;

5) Finally, we asked the interviewees to report the benefits perceived and the hidden challenges related to contribute to OSS project as part of a Software Engineering course. We also asked them to indicate the importance of the instructors' involvement in OSS initiative (extra-classroom), if any. Finally, as ask the interviewees to suggest improvements in the courses.

C. Interview Analysis

The interview analysis comprehends four steps:

1) Familiarizing with the data: At this stage, we read and re-read the interview transcript several times to get acquainted with particular terms that we were not yet familiar. When needed, we also searched on forums and mailing lists to better familiarize with the terms.
2) Initial coding: In this step, we added codes, that is, labels that could express the meaning of the excerpts of the interview that had appropriate actions or perceptions. The initial codes are considered temporaries since they still need refinement. The codes were identified and refined throughout all the analysis.
3) From codes to categories: Here we already had an initial list of codes. We then begin to look for similar codes in the data. We grouped the codes with similar characteristics in broader categories. Eventually, we also had to refine the categories found.
4) Categories refinement: Here we have a potential set of categories. We then search for evidence that supported or refuted the categories found. We also renamed some categories to describe the excerpts better there. The final categories are present throughout Section III.

We conducted the interview analysis in pairs followed by conflict resolution meetings. During the meetings, at least one more co-author was present to mediate the discussion. The categories discussed in this paper are the result of these consensus meetings.

D. Summary of the participants

Table I shows a summary of the students that participated in our study. All interviewees are Brazilian, and their interviews were conducted in Portuguese and later translated into English. We refer to the participants as P1–P21. The participants P9, P20, and P21 took two courses those leverage contributions to OSS in two different universities (U3 during their undergraduate studies and U4 during her graduate studies). In this case, we grouped her perception according to the university which they took the course. As one can see, seven out of the 21 participants are female, and seven participants took the courses while working towards their Masters or PhDs. The majority of the participants (16 of them) did not have previous experience in contributing to OSS projects before the course. The participants were required to contribute to OSS during part of the term of the course, which varied from 2–3 weeks up to 2 months. On average, the participants have 27.8 years old.

III. Results

In this section, we present our main results grouped by the research questions.

A. RQ1: What are the strategies used to choose the open source projects that students would contribute to?

Among the strategies used to choose OSS projects, we highlight (i) the freedom to choose OSS projects (9 occurrences), (ii) the instructors' indication (8 occurrences), and (iii) the previous knowledge about the OSS community (4 occurrences). Here we discuss these strategies.

1) Freedom to choose open source projects: According to eight students, the instructor gave total freedom to students to choose the OSS projects they would like to contribute. For instance, P1 mentioned that the instructor "*[...] gave us the freedom to choose the project we wanted to contribute to*". Participant P3 also mentioned that "*the instructor did not impose any restrictions related to the open source project*". Still, the participant P11 affirmed that "*[...] the instructors provided the necessary background and made students free to choose any interesting open source project*". Also, P8 described that "*we could choose any open source project [. . .] we chose an open source project initially created (in the past) inside the university; therefore, we could be closer to the maintainers*".

We also observed other scenarios in which the instructor gave partial freedom to the students choose the projects. In this case, the instructor had a set of guidelines that the students should follow when looking for representative OSS projects. As an example, P5 mentioned that "*the criteria for choosing the OSS project were the simplicity of the project, previous experience with the programming language, and the affinity with the project*". In another institution, P9 cited such approach in the course they took: "*generally speaking, the only strong requirement was that the project should be active. Even better if the project could be related to our master's research*".

2) Instructors' indication: For some courses, we observed that the chosen projects were based on the instructors' indication. Five students stated this, as P2 highlighted: "*Firstly, the instructor chose the OSS project for us*". However, we also observed cases in which the instructor had a previous list of potential projects, and the students had to decide which one from that list interest them. P5 reported that "*the students had to choose between GNOME or KDE*". The instructors' participation in OSS initiatives also appeared as a relevant factor for choosing a project, as P9 pointed out: "*before the course started, the instructor already filtered several maintainers from*

TABLE I
DEMOGRAPHICS INFORMATION OF THE PARTICIPANTS.

#	Age	Gender	University	Level of the course	Current occupation	Previous exp. with OSS?	Year of the course
P1	27	Female	U2	Undergraduate	Software Developer	No	2013
P2	23	Female	U5	Undergraduate	Software Developer	No	2015
P3	25	Female	U5	Undergraduate	Graduate Student	No	2013
P4	24	Male	U3	Undergraduate	Lecturer (temporary)	Yes	2013
P5	25	Male	U5	Undergraduate	Software Developer	Yes	2016
P6	26	Male	U3	Undergraduate	Software Developer	Yes	2015
P7	26	Male	U2	Undergraduate	Software Developer	No	2013
P8	24	Male	U2	Undergraduate	Software Developer	No	2013
P9	27	Male	U3/ U4	Undergraduate / Graduate	Graduate Student	No	2014/2016
P10	25	Female	U3	Undergraduate	Consultant	No	2015
P11	30	Male	U3	Undergraduate	Graduate Student	No	2013
P12	34	Male	U1	Graduate	System Analyst	No	2012
P13	33	Male	U1	Graduate	Entrepreneur	Yes	2012
P14	31	Male	U2	Undergraduate	Software Developer	Yes	2013
P15	23	Male	U5	Undergraduate	Graduate Student	No	2016
P16	22	Male	U5	Undergraduate	Graduate Student	No	2016
P17	36	Female	U1	Graduate	Graduate Student	No	2012
P18	23	Female	U5	Undergraduate	Graduate Student	No	2016
P19	50	Female	U1	Graduate	Faculty	No	2012
P20	24	Male	U3/ U4	Undergraduate / Graduate	Graduate Student	No	2016/2018
P21	26	Male	U3/ U4	Undergraduate / Graduate	Graduate Student	No	2013/2018

some consolidated OSS projects. After that, they observed who would have availability to give some support to the students".

Finally, P6 indicated a collaborative way for choosing the OSS projects to contribute to: "*The instructor selected the list of enrolled students at the beginning of the semester and grouped the students based on their previous programming experience. If the student did not have the necessary experience, they gave a challenge to that student. They also asked the students from the lab to assist the class. All the students gave their opinions about the selected projects. Finally, the instructor checked the background of the students. If, for example, the student had a good background in C, then the student could work with this project here[...]* ". Still, to assess students' experience, the instructor also asked students to fill a questionnaire at the beginning of the semester. In this questionnaire, the students should indicate three OSS projects from the initial set of projects that the instructor curated. The goal is to have teams with balanced programming experience. The students with good programming background played a "coach" role in the team (as in the XP agile method). In this case, the project was chosen using a set of criteria that the instructor defined after evaluating students' background.

3) Prior knowledge about the open source community: We also identified some cases in which the OSS projects were chosen based on prior knowledge about the community responsible for that given OSS project. This prior knowledge about the OSS project and its community could either be from the instructor or the students. As an example, participant P4 mentioned that one student "*chose an OSS project that they were already familiar with*". Moreover, P1 mentioned that "*[the instructor] already knew the maintainers that created the game we were interested in contributing to. Therefore, since we could easily get in touch with them, we decided to work on this project.*" In two cases, the instructor had the previous contact with the OSS community. One of these cases was pointed out by participant P4: "*the instructor chose several projects that they had good contact with the community [...] from varied programming languages and domains*". Finally, P10 discussed that: "*the instructor chose some projects that they found interesting, or projects that they had a contact from within*".

RQ1 Summary: Among the strategies employed to choose OSS projects, some instructors gave total or partial freedom to the students to choose whatever project they want to work. Other instructors provided some criteria that the students should meet, while other instructors invited students and maintainers to choose the project collaboratively.

B. RQ2: How do students choose their tasks?

After choosing the project, the next step is to choose the task(s) that students should work on during the course. Participant P5 bring the attention to the fact that, before look for tasks, it is essential to "*study the OSS project*". Afterward, P5 also mentioned that it is important to "*pick a peripheral task, and avoid tasks related to the core of the project*". Still, P5 indicated that one possible task to get acquainted with the project is to "*pick a bug and try to reproduce it*".

Several OSS projects studied in the courses were hosted on social coding platforms, such as GitHub and Gitlab. In this context, some students mentioned the fact that, on GitHub, the "*issue tracking system has a list of potential tasks that students could search for*". Based on this list of issues, P6 commented that one particular OSS project created labels to properly identify "*issues that are suitable for students work on*". P10, P16, and p18 also indicated that the use of labels in issues assisted students to search for a possible task. P10 noted that "*the project had issues with that determined it was easy*

or challenging. The maintainers of this project also assisted in setting up the project and guided the students when needed."

The search for tasks in issues in the issue tracking system was a recurrent topic reported by other students working on different projects in other institutions. For instance, P7 reported that "*the default way was to search for tasks in the issue tracker*". P18 complemented that they "*searched for tasks and bugs using 'newcomers' keyword in the issue tracking system*". Interestingly, P16 reported that some issues tagged as "newcomers" were straightforward to solve: "*in these issues, it would be easier to the maintainer to solve the problem rather than create an issue properly describing the problem.*" P16 hypothesized that maintainers opted to create the issue (rather than solving it) as a way to ease the onboarding process of external members to the OSS project.

In order to evaluate the feasibility of implementing the task in the context of a course, P7 indicated that "*the students tried to communicate with the OSS communities to see if the planned work made sense, or whether it would be doable.*" Similarly, P8 reported that they "*searched for open issues*". Afterward, P8 "*picked some issue that they thought they could solve and that would be interesting to contribute to the project.*" However, P16 also highlighted that one should be aware of the time taken to choose a task: "*the project we were working on had too many issues reported [...] we spent too much time looking for an appropriated task*".

Finally, we also observed a collaborative way of choosing tasks to students work with, as P11 reported: "*We (instructor, students, and the project maintainer) had meetings to discuss potential tasks to work. The maintainer suggested some tasks that are of great interest to the project, the students discussed what they wanted to do, and the instructor discussed about whether the task was feasible to the scope of the course*". Additionally, participant P9 corroborated with the collaborative strategy, as they pointed out that "*the definition of the tasks was very democratic, using agile methods. The students sat together with the instructor one day of the week, opened the issues and discussed what could be done*". Such collaborative strategy varied between the projects, as well as the level of involvement of the students. When students were still getting acquainted with the projects, the instructor participated in planning meetings to make sure students could have something done during the first month of the course. This planning was essential to motivate students to keep contributing throughout the course.

Summary of RQ2: Before start looking for some task to work on, students have to find the time to get acquainted with the OSS project. Afterward, looking for a task in the issue tracker was recurrently commented. Issues with labels that indicate the complexity of the task could ease the decision regarding what task to work on. If maintainers are accessible, instructors, students, and maintainers could collaboratively find what to work on.

C. RQ3: What is the nature of the students' contributions?

Table II presents the OSS projects that students contributed to, as well as quantitative information about these projects. In this work, we consider a contribution as an accepted *commit* made at the official OSS repository.

Despite being sent to the OSS project and have been evaluated by the OSS maintainers, the participants P5, P12, P15, and P19 did not have their contributions listed in Table II because they were not accepted. On the other hand, some projects received contributions from more than one student (for instance, project `analizo`). Moreover, some participants contributed to more than one project during the same course, which is the case of participant P9, which contributed to both `prezento` and `kalibro`, during her undergraduate studies, and contributed to project `yosys` during her graduate studies. Still, P21 contributed to `elixir-bench`, which is an umbrella for many other OSS. According to him, he contributed to six OSS projects under `elixir-bench`. Therefore, the columns "# Commits", "# Contributors", and "# Lines of Code" in this case refer to the `elixir-bench-runner` project, which is the most popular project (in terms of number of stars) under `elixir-bench`.

As one could see at Table II, the projects varied concerning size and programming language. The projects' domain also varied from games to mobile platforms, source code analyzers, web frameworks, among others. Some of the projects were created and are maintained by contributors in the same university that student took the course (e.g., the `catch-the-pigeon` project); while other projects are well-known from the software development community (e.g., the `cakephp` project). Most of the students had 1–2 contributions accepted; P21 was an outlier: he had 30+ contributions accepted and became a maintainer while taking the course. In total, we found 89 contributions.

In order to assess the nature of the software development activity, we employed the definition of Hattori and Lanza [9], which group the contributions in four broad groups. They are:

1) **Forward engineering**, for instance, adding new features;
2) **Reengineering**, for instance, refactoring activities;
3) **Corrective**, for instance, fixing bugs;
4) **Management**, for instance, updating documentation.

For each participant, we search for the contributions performed. To find such commits, we asked students the links of their contributions. In the case which the students did not recall the links, we asked them their usernames, so then we could search for their contributions in the commit history of the studied OSS projects. We used the commit message to categorize the intention of the code changes. When possible, we tried to match the commit intent with the purpose described in the interview.

We used a previous list of keywords introduced by Hattori and Lanza [9] to support the grouping procedure of the contribution' intent. After applying this procedure, we observed that the majority of the contributions were related

TABLE II
THE LIST OF OSS PROJECTS THAT STUDENTS CONTRIBUTED TO. THE COLUMNS # COMMITS AND # CONTRIBUTORS REPRESENT THE TOTAL NUMBER OF COMMITS PERFORMED AND THE TOTAL NUMBER OF CONTRIBUTORS WHO AUTHORED THESE COMMITS. LINES OF CODE (LOC) WERE CALCULATED WITH THE `CLOC` UTILITY, CONSIDERING COMMENTS AND BLANK LINES. THE "PL" COLUMN MEANS THE PROGRAMMING LANGUAGE MOST USED IN THE PROJECT, CALCULATED WITH THE `LINGUISTIC` UTILITY.

#	Project	# Commits	# Contributors	# Lines of Code	PL	Domain
P1	catch-the-pigeon	197	11	7K	Java	Android game
P2	jabref	12K	143	138K	Java	BibTeX manager
P2	gnome-music	2K	198	31K	Python	Music player
P3	L.Office Impress	1K	39	18k	Objective-C	Office suite
P4	noosfero	15k	126	631K	JavaScript	Content Management System
P6, P9	prezento	1.7K	35	13K	Ruby	Web interface tool
P7	diaspora	19K	338	151K	Ruby	Social network
P8	amadeus	3K	13	114K	Python	Online learning system
P9	kalibro	1.2K	12	7K	Ruby	Source code analyzer
P9	yosys	3.8k	54	121K	C++	Verilog RTL synthesis suite
P10	radar-parlamentar	2k	56	35K	Python	Web crawler
P10	gestorpsi	2k	14	103K	Python	Clinic organization system
P4, P11	analizo	1.1k	18	7K	Perl	Source code analyzer
P13	cakephp	35k	505	184K	PHP	Web framework
P14	liferay-portal	264k	478	5,262K	Java	Web platform for building business
P16, P18	polari	2K	107	40K	JavaScript	Internet Relay Chat
P17	joomla!	14K	78	320K	PHP	Content Management System
P18	teammates	16K	370	713K	Java	Education management tool
P20	spark	22K	1K	739k	Scala	A cluster computing system for Big Data
P21	elixir-bench	32	4	750	Elixir	Benchmarks for Elixir projects

to **Forward engineering**; eleven participants reported that introduced new features. For instance, in the interview, the participant P4 mentioned that her team made "*one contribution to the LibreOffice Impress repository, to change slides using a smartphone. There was no such feature, and we added*".

Eight participants performed contributions with **Corrective** intentions. For instance, participant P2 observed a bug during "*the process of importing to a database. It worked only when using MySQL, but did not work when using Postgres.*" The participant confirmed the bug with the OSS community and implemented the fix. Moreover, only four participants performed contributions related to **Reengineering** intentions. Among the examples, we highlight the contribution made by participant P8, which stated in the commit that they "*refactored some layouts that were not quite appropriated, according to the Android best practices*".

Finally, only one participant mentioned a contribution related to **Management** intentions. This participant mentioned that they reported bugs or added labels on issues as part of her contributions. It is important to note that five participants followed an agile method approach throughout the course, as P4 highlighted: "*We followed the Scrum process. We had sprints, user stories (and planning poker to estimate the stories), etc.*", which was fostered by the instructor. The instructor also used pair programming to improve the transfer knowledge between the participants within a team.

Next, we also investigate the code churn of the contributions performed. Figure 1 presents the distributions of these three variables (i.e., files edited, lines added and removed). When analyzing the number of additions and deletion in this data, we could found several outliers. For instance, one single contribution added a total of 2,711 lines of code and also removed 2,589 lines of code[1]. This contribution changed the layout of an Android game; then it was necessary to apply this change in 39 different files. On average, the contributions that students made edited 12.5 different files, added 56 lines of code and removed seven lines of code. This finding is in contrast to the current understanding of contributing best practices, which suggest that pull-requests should be small [10]. However, small contributions were also found. The first quartile of contributions have four lines of code added and one line removed. For instance, contribution 114[2] performed in the project `prezento` made one single change: updated one figure in a web page.

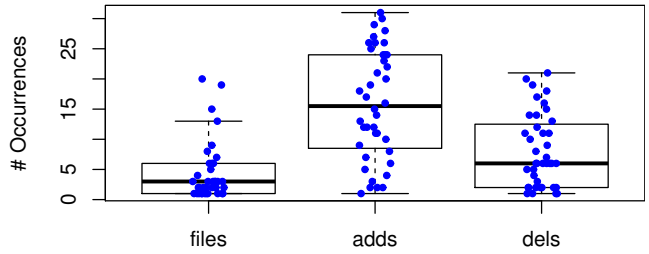

Fig. 1. The distribution of the contributions that students made to the OSS projects. We removed the outliers from the figure to ease visualization.

[1] https://github.com/rodolfoasantos/catch-the-pigeon/commit/51607083
[2] https://github.com/mezuro/prezento/pull/114/files

Summary of RQ3: Most of the contributions found at the repositories had forward engineering intentions (e.g., adding new features). Corrective intentions (e.g., fixing a bug) were also common. Although small contributions are frequent, on average, a student contribution comprehends 56 additions in 12 different files — much higher than the current understanding of contributing best practices.

D. RQ4: What are the benefits?

Regarding the benefits of participating in such kind of courses, we observed three main categories: (1) to participate in a real project, (2) to improve their resume, and (3) to become an active contributor.

1) Working on a real project: The students felt motivated and challenged to participate in a real project. Seven participants shared this perception. As an example, participant P2 mentioned that "*it increases your confidence. You collaborate in a real project; anyone could see and use your contribution [...] it is a real-world experience.*" P4 complemented that "*it is a great opportunity for a student. First, you can contribute to an existing non-trivial project. Second, it gives you a chance to meet OSS communities and understand that contributing to OSS projects is not a nightmare.*". Moreover, P14 mentioned the "*importance of the philosophical aspects of contributing to OSS projects*". Further, P7 shared an altruism notion that "*you become happier when, besides having learned, you contributed a little bit with a project that is being used elsewhere*". The participant P12 also mentioned that "*you could understand how does the project's ecosystem survive, and how you could work in a network of distributed developers.*" Along these lines, P9 mentioned that "*you could see that a single person does not do an OSS project; instead, it is a collaborative effort*".

2) Improving the resume: Improving the students' resume was perceived as one benefit by three participants. According to P8, the experience acquired in the course "*was the start point onboard in the OSS world*". P6 reported that "*Today, in the company that I work for, I am the most skilled person in version control systems, which is mostly due to the course I took*," and complemented saying that "*it was an immeasurable benefit to arrive in a job interview and state that I have contributed to OSS projects*". Moreover, P10 added that the course was an opportunity for "*[...] build a portfolio, which is essentially your contributions, and you can bring that to a job interview [...] many students that liked the course and succeed contributing during the course were accepted in graduate programs and received job offers from big companies in Brazil and abroad. A lot of it was because of the portfolio that we are building while studying*".

3) Becoming an active member: Regarding the continuity of the contributions to OSS software projects after the course, although some participants did not keep contributing, we observed that eight students become active OSS contributors. In particular, participant P11 mentioned that "*after the course I spent quite some time looking for ways to get involved in some OSS projects. I started by sending random pull-requests to projects hosted on GitHub that I found interesting. I did that until I found one specific project that I kept contributing until today*". Still, P9 mentioned that they became a Linux contributor. "*Nowadays I am very focused on Linux [...] I also contribute to Debian, but I do not implement features in there, I work with the package management system; I have two Debian packages, and I am preparing other three*". Finally, P21 reported that the course helped him to participate in the Google Summer of Code (GSoC) program, which he had the opportunity to become even more active. These three participants (P9, P11, and P21) did not have previous experience contributing to OSS projects before the course.

Summary of RQ4: Working in a real project was one of the most positive impressions that students reported, which improve not only students' skills but also their self-confidence. Some participants received job opportunities in Brazil and abroad, while others participate in GSoC. Other students became active contributors in well-known, non-trivial projects (e.g., Debian Linux), even if no previous experience contributing to OSS projects

E. RQ5: What are the challenges?

During the interviews, the participants mentioned some facts that hinder their attempt to contribute to OSS projects. After analysis, we observed that the main challenges are (1) the source code complexity, (2) to interact with the OSS community, (3) to understand and set up the software development environment, and (4) lack of time to contribute.

1) Source code complexity: The participant P2 reported that "*[...] the difficulty was to understand the source code structure*". P3 shared a similar concern, when they mentioned that " *[...] we had to analyze the whole project [...] until you get acquainted with it. It is a hard process*". Participant P4 mentioned that "*[...] it is always hard to understand the software architecture and the source code*". Participant P5 stated that "*the challenge that I face was with the source code itself because it did not follow any convention [...] it looked like several shuffled cards, which we were unable to sort out.*" The student P8 mentioned that it was "*hard to understand how the source code was organized [...] it was also hard to follow the conventions used in each project*". P21 shared a similar perception: "*I thought I knew Python, but when I joined the OSS, I perceived I knew nothing about Python*". Finally, participant P7 reported that the main challenge was "*[...] to get acquainted with the project source code and guidelines*".

To mitigate this kind of challenge, the instructor grouped students in a way that one student that was facing more challenges could pair with someone more comfortable with the project. Still, some instructors got in touch with the maintainers to solve some open questions. Finally, during some classes, students were also motivated to study the source code collaboratively, using techniques such as *coding dojo*.

2) Interactions with the open source community: During the process of implementing the tasks, often students had to interact with the OSS community. According to P3, "*it is hard to interact in a mailing list since you do not know who is who, and who will answer you [...] a colleague of mine sent a message to a mailing list and got a very unpolite reply*". Another challenge, reported by P4 and P9, is related to understand "*the organization of the community itself*". P5 also mentioned that "*the community is not active anymore*", which makes harder to have their questions answered. Finally, the last report along these lines was made by P11 that affirmed that "*[...] it is hard to have the first contact with someone from inside the community [...]*".

3) Setting up the environment: Some respondents mentioned that it was a challenge to understand and set up the environment of the studied projects. According to participant P1, "*my team had previous experience with Linux, but I did not*". Participant P5 reported that "*[...] the version control system was unknown for us*". Moreover, P7 indicated that "*[...] it is not uncommon to have a headache to set up the project*". Still, P9 also mentioned it was "*[...] a hard time to learn git [...]*". P17 shared a similar concern: "*I had to send my changes through git command line. I had neither experience with git or command line*".

4) Lack of time to contribute: Due to the hard time to get acquainted with the OSS project, some participants mentioned that the duration of the course was not sufficient. Participant P4 mentioned that the course could be divided into two parts: the first one which focus was on getting acquainted with OSS projects and the contribution model, and the second one which the students would contribute to an OSS project. According to P4, with two courses, the students would not need to rely on the maintainers to answer their questions. Similarly, P1 reported that the course could be divided into two semesters.

Summary of RQ5: Regarding technical challenges, students had a hard time to understand the code structure and how to set up the project. Another challenge was to get involved with the community (mailing lists do not help much since you do not know who is who). The short duration of the course was a challenge.

IV. Further Analysis

In this section, we provide additional discussion on the data presented in the previous sections.

Previous experience with OSS software development is not mandatory. As one could observe at Table I, 16 out of the 21 participants did not have previous experience in contributing to OSS software. However, from these 16 participants, 12 succeeded in transposing the first contribution barriers and had their contributions accepted. In particular, participant P21 performed 30+ contributions.

Implementations that go beyond Java and C++. Even though some projects were developed using traditional programming languages (such as C++ and Java), we could observe that some students contributed to projects written in programming languages that are not so common in software engineering curriculum (such as Perl, Elixir, and Objective-C). Indeed, P21 mentioned that they purposefully chose one OSS project in a programming language that they did not master as a way to learn this programming language. This finding demonstrates the feasibility of expanding the curriculum to cover additional programming languages.

Students' self-assessment. Generally speaking, the participants reported a good performance in the courses. Only P9 mentioned having regular-to-good performance, while participant P5 mentioned having insufficient performance. According to P5, their performance "*would be better if I chose a more active OSS project.*" On the other hand, we could see that the participants well received the process. P11, for example, reported that "*every course in undergraduate level should leverage OSS contributions [...] If an Instructor needs to talk about one kind of programming implementation, the Instructor could discuss real implementations using an OSS project as an example*".

Patches or pull-requests? Even though the majority of the contributed projects is hosted in social coding platforms such as GitHub, we observed that in one of the studied projects (`L. Office Impress`), the contributing process occurred sending patches to the mailing list. In this particular project, the students manually had to generate the patch using a command line `git` utility and, still, discuss this patch in another coding platform (Gerrit, in this case). Although slightly different from the modern contributing practices [10], the contributions sent as patches were accepted.

The delay in processing a contribution. Although already discussed in other studies (e.g., [11]), OSS communities tend to take longer to process contributions when the author is not an active member of the community. The participants of this study also observed this assertion. For instance, P9 is still waiting for a response to the contributions made in 2016. P9 mentioned that *"so far, two of my four pull-requests are still open [...] they did not respond to the pull-requests yet"*. P1 also noticed similar behavior in her colleagues' work: "*some colleagues had swift feedback and succeeded to contribute [...] Some other colleagues did not have the same luck. Some of them did not receive any response during the period of the course. I still do not know if they have any contribution accepted.*" P16 still has unprocessed pull-requests.

Size of the contributions. Recent studies suggest that contributions to OSS projects tend to be small in nature [11], [10], [12]. Differently, our sample of contributions has a high rate of edited files and lines added/removed per contribution. Since the current recommendation is to perform small and self-contained contributions (for instance, including automated tests), which ease the code review process on the integrators' side. Consequently, the size of the students' contribution might indicate a potential lack of experience in the software development process. Although this behavior might be expected (i.e., our subjects are students, and some of them did not have either professional or OSS experience before the course), this fact

corroborates the importance of this kind of approach in which students acquire such practical experience before applying for positions in industry.

Contributions with test cases. We found that 37 out of the 89 contributions performed were submitted with test cases. To identify test cases, we observed whether the diff code has any method that ends or starts with "test" and if this method resides in a file that ends or starts with "test". `noosfero` was the project that received the most contributions with test cases (7). This finding is in contrast with recent studies that suggest that many OSS projects do not accept (or even review) contributions that do not come with test cases [13].

Good community practices for OSS communities. Some participants also mentioned best practices that, if incorporated in OSS communities, could ease the process, students in particular. For example, P5 indicated that "*[...] sometimes the project has some documentation, but does not have any developer-oriented documentation*". Examples of such developer-oriented documentation include guidelines on how to install and set up the project. P13 corroborated with the importance of having good documentation in OSS projects: "*we studied why one project is more successful than others, and part of it is due to very well-done documentation*". Another aspect also raised by P5 is "*lack of labels on issues to identify whether the issue is for beginners or advanced developers*". P5 went further and mentioned that the OSS project should have, indeed, documentation in a way that newcomers could quickly contribute to newcomer-related issues, whereas advanced programmers could contribute to the core of the project.

The instructor involvement in OSS initiatives. Some students highlighted the importance of having an instructor involved in OSS initiatives. In particular, P3 indicated that such involvement "*was essential, since the instructor had more practical experience. Therefore, when students were blocked in a task, the instructor could either properly help them or find someone who could provide help.*". Moreover, participant P5 indicated that, due to academic research conducted by the instructor, the students were already aware of some of the challenges that they might face: "*our instructor studied ways to understand drop out causes in the OSS project. We faced some of those challenges.*" Still, participant P10 stressed that "*if the instructor is not prepared, there is no chance to have such dynamic. It has to be an instructor who understands and is passionate about OSS software.*" Finally, participant P5 mentioned that besides being involved with the OSS community, it is equally important to "*engage students to follow the right path*".

V. Implications

This work has implications for several stakeholders. We discuss two of them next.

Software Engineering instructors. As we found in the previous sections, contributing to OSS communities was perceived as beneficial by many students. As a potential consequence, many other instructors could experiment with this approach in their courses. For instructors that already made the shift to courses that leverage OSS contributions, we believe they can improve their agenda by, say, challenging students to contribute to projects written in programming languages that they do not master or fixing bugs associated with more critical issues. Similarly, instructors that teach testing courses could foster students to create test cases for contributions that did not have tests (37 out of the 89 contributions go without test cases). Finally, to better understand the challenges that students face, instructors themselves could try to contribute to OSS project or to participate in OSS initiatives.

OSS Project Maintainers. OSS maintainers could get inspired by some of the findings of this paper. For instance, maintainers could create issues that could be solved in a short amount of time by someone with little to no previous experience in the project. Although it might be tempting to fix the problem rather than creating an issue describing the problem, this approach could ease the life of potential new contributors. OSS maintainers could also put a particular effort in the documentation. For instance, README files could state which maintainers are available to support students to onboard the project. Similarly, since many students had a hard time trying to understand the source code, maintainers could improve the documentation by better describing their architecture, conventions, how to set up the project, etc.

VI. Limitations

As an empirical study, this one also has many limitations and threats to validity.

First, our study is limited by the number of students that participated in our study. Although we conducted 21 interviews, we still believe our findings might not easily generalize. Moreover, the findings of this study are limited to the interviews conducted in the five studied institutions where our participants took the courses. The perceptions of other students that took similar courses in other institutions are still unknown. Similarly, this work is limited to software engineering courses that took advantage of OSS contributions as part of their learning model. The use of OSS contributions in other courses — which could lead to different findings — was also not explored. Another limitation is related to when our participants took the courses. Some of them took the courses in 2012 and 2013, so their reflection might not be accurate. To mitigate this bias, we interviewed participants from different universities that took the courses in different time windows. Finally, although enriched with some quantitative data regarding the size of the contributions performed, this work is mainly based on the interviews conducted. Therefore, it is limited to our understanding of what the participants said. To mitigate threats hidden in qualitative analysis, after we transcribed the interviews, we analyzed each interview in pairs, followed by conflict resolutions meetings. When needed, third research was involved in the conflict meeting to drive consensus.

VII. Related Work

Several recent efforts studied the dynamics of bringing OSS projects in the context of a classroom [14], [15], [16], [17], [18]. Smith and colleagues [15] focused on selecting the most appropriate projects for students work. The authors claimed that, due to the short duration of a traditional software engineering course (4–5 months), the instructors should not select large or complex OSS projects, since the students would face many difficulties to contribute significantly to these projects throughout the course. In our work, we perceived that many OSS project that received contributions are large ones (on average they have 454.4K lines of code).

Similarly, other works suggest that the OSS projects should not be rather small or naive since the students would not have the chance to put into practice important software engineering principles. Morgan and Jensen [14] detailed the experience of teaching a software engineering course based on OSS projects. The authors observed that the size of the project (Ubuntu, in this case) was a real obstacle for students to contribute (for instance, some students were unable to find a simple yet exciting task to work on). Likewise, Buchta and colleagues [16] reported their experience in teaching software maintenance and evolution aspects in a software engineering course. The authors reported a non-trivial setup effort of about 60 hours before the course takes place. However, the authors did not discuss the potential benefits or challenges related to this OSS model. Holmes and colleagues [19] reported the lessons of their Undergraduate Capstone OSS Projects (UCOSP). In this capstone project, the studied OSS projects should (1) have an active issue tracker, (2) use a version control system, and (3) have a well-defined code review process.

To the best of our knowledge, the works of Holmes et al. [3] and our previous work [6] are the ones closest to this current study. Holmes et al. [3] also studied the students' perception regarding the adoption of OSS contributions. The authors observed that students' took advantage of the opportunity to apply their skills in real tasks, from real projects, while receiving real feedback from project maintainers. Still, the authors reported that contribute to the real project provides a greater understanding of software engineering aspect when compared to more traditional means. One of the main differences between our studies is that Holmes et al. [3] expect students to contribute to OSS as part of their capstone projects, while in our case, the students should contribute as part of a course. This point is particularly important because students might be more challenged to contribute during a course since the term of a course is often shorter than a capstone projects. Also, contributing to OSS is only part of the assignments of a course; in some cases, students still have exams and other assignments. Moreover, capstone projects often occur in the final years, when students are more skilled. In our interviews, we perceived that some of the students took courses in their second year. Further, although part of our work can be seen as complementary to Holmes' works, our work is unique in the sense that we studied not only the benefits and the challenges but also the nature of the contribution and other students' attitudes.

In a previous study [6], we investigated the challenges, benefits, and the approach used to select OSS projects looking at the instructors' perspective. Indeed, we were inspired by their work to conduct our work. Differently than this previous work [6], which interviewed instructors, we interviewed students that participate in such kind of courses. Our current work confirms — by investigating three similar research questions — and expands the previous one [6] since it brings to the discussion the students' perceptions, which was not well understood.

VIII. Concluding Remarks

Software engineering instructors are taking advantage of OSS projects as part of their evaluation model, and recent work exposed many benefits perceived with these courses. However, such studies focused on the instructors' perspective, while the students' perspective was still unclear.

In this work, we conducted 21 semi-structured interviews with students that participated in software engineering courses that took advantage of OSS projects as part of their evaluation model. Despite the known challenges related to finding a typical project and a suitable task to solve in the short period of the course, many students reported positive impressions regarding the course. The majority of the students also positively self-evaluated their performance in the course. Moreover, we observed that the instructors' participation in OSS initiatives (extra-classroom) are notoriously important for a better dynamism between instructors, students, and OSS communities. Finally, we also observed that the majority of the students' contribution was intended to add new features in the OSS projects studied, although corrective contributions were also common. On average, a student's contribution added 56 lines of code in 12 different source code files.

For *future work*, we plan to conduct a large-scale study in OSS repositories to investigate and categorize students' participation in the context of a course. Still, we also plan to conduct interviews with OSS maintainers and mentors, triangulating the findings of this and previous studies, to confirm or refute known findings in the literature.

Acknowledgments

We thank the students for participating in the study and the reviewers for they useful feedback. This work is partially funded by PROPESP/UFPA, Northern Arizona University, CNPq (#406308/2016-0 and #430642/2016-4), and FAPESP (Grant #2015/24527-3).

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] R. Holmes, M. Allen, and M. Craig, "Dimensions of experientialism for software engineering education," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training*, ser. ICSE-SEET '18. New York, NY, USA: ACM, 2018, pp. 31–39. [Online]. Available: http://doi.acm.org/10.1145/3183377.3183380

[4] 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.

[5] 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.

[6] G. Pinto, F. Figueira Filho, I. Steinmacher, and M. Gerosa, "Training software engineers using open-source software: The professors' perspective," in *30th Conference on Software Engineering Education and Training (CSEET)*, 2017, pp. 117–121.

[7] G. Pinto, I. Steinmacher, and M. Gerosa, "Leaving behind the software history when transitioning to open source: Reasons and implications," in *Proceedings of The 14th International Conference on Open Source Systems*, 2018.

[8] C. Chavez, A. Terceiro, P. Meirelles, C. Santos Jr, and F. Kon, "Free/libre/open source software development in software engineering education: Opportunities and experiences," *Fórum de Educação em Engenharia de Software (CBSoft'11-SBES-FEES)*, 2011.

[9] L. P. Hattori and M. Lanza, "On the nature of commits," in *23rd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 2008, pp. III–63.

[10] G. Gousios, M. Pinzger, and A. van Deursen, "An exploratory study of the pull-based software development model," in *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, 2014, pp. 345–355.

[11] G. Pinto, L. F. Dias, and I. Steinmacher., "Who gets a patch accepted first? comparing the contributions of employees and volunteers," in *11th IEEE/ACM International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE@ICSE 2018, Gothenburg, Sweden, May, 2018*, 2018.

[12] G. Gousios, A. Zaidman, M. D. Storey, and A. van Deursen, "Work practices and challenges in pull-based development: The integrator's perspective," in *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, 2015, pp. 358–368.

[13] I. Steinmacher, G. Pinto, I. Wiese, and M. A. Gerosa, "Almost there: A study on quasi-contributors in open-source software projects," in *40th IEEE/ACM International Conference on Software Engineering*, 2018.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] A. Sarma, M. A. Gerosa, I. Steinmacher, and R. Leano, "Training the Future Workforce Through Task Curation in an OSS Ecosystem," in *2016 24th ACM FSE*, ser. FSE 2016. ACM, 2016, pp. 932–935. [Online]. Available: http://doi.acm.org/10.1145/2950290.2983984

[19] 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. New York, NY, USA: ACM, 2014, pp. 321–324. [Online]. Available: http://doi.acm.org/10.1145/2591062.2591160