electronics

Article

Leveraging Final Degree Projects for Open Source Software Contributions

Juanan Pereira

Department of Computer Languages and Systems, Computer Science Faculty, University of the Basque Country, UPV/EHU, 20018 Donostia, Spain; juanan.pereira@ehu.eus

Abstract: (1) Background: final year students of computer science engineering degrees must carry out a final degree project (FDP) in order to graduate. Students' contributions to improve open source software (OSS) through FDPs can offer multiple benefits and challenges, both for the students, the instructors and for the project itself. This work reports on a practical experience developed by four students contributing to mature OSS projects during their FDPs, detailing how they addressed the multiple challenges involved, both from the students and teachers perspective. (2) Methods: we followed the work of four students contributing to two established OSS projects for two academic years and analyzed their work on GitHub and their responses to a survey. (3) Results: we obtained a set of specific recommendations for future practitioners and detailed a list of benefits achieved by steering FDP towards OSS contributions, for students, teachers and the OSS projects. (4) Conclusions: we find out that FDPs oriented towards enhancing OSS projects can introduce students into real-world, practical examples of software engineering principles, give them a boost in their confidence about their technical and communication skills and help them build a portfolio of contributions to daily used worldwide open source applications.

Keywords: open source; final degree projects; software engineering; computer science education

Citation: Pereira, J. Leveraging Final Degree Projects for Open Source Software Contributions. *Electronics* **2021**, *10*, 1181. https://doi.org/10.3390/electronics10101181

Academic Editor: Claus Pahl

Received: 14 April 2021
Accepted: 13 May 2021
Published: 15 May 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

During the last 20 years, Software Engineering (SE) instructors have proposed the study and development of open source/free/libre software (abbreviated as OSS, FLOSS, or FOSS) projects to introduce students into real-world examples of SE principles, concepts and practices [1–3]. Mature OSS projects are considered more realistic than examples found in textbooks or developed by teaching staff [2,4]. These projects are usually complex and large, with many developers involved, responding to real-world needs and problems and are built following software engineering best practices. Due to their open nature they also provide opportunities for studying not only their source code but also a whole set of software-related artefacts developed during their life cycle: requirements, design, and project management documentation, configuration management-related artefacts (version control, bug tracking and build management), testing, deployment, quality control and, specially, evolution and maintenance of software. These projects also allow students to interact with external, distributed development teams, discussing designs, problem solutions or future directions of the projects in public forums, mailing lists, bug tracking systems or code review panels.

SE instructors, researchers and practitioners have at their hands a plethora of open source projects neatly organized in popular social code sharing repositories such as GitHub, GitLab or Bitbucket that help them to analyze, study and contribute to OSS projects. It is not surprising then that many researchers had resorted to this kind of software to teach and illustrate software engineering concepts to their students in real-world scenarios: Nascimento et al. [5] studied the use of OSS in Software Engineering Education (SEE) from 1998 to 2012 and Brito et al. [1], updated the mapping study with findings from the 2013–2017 period.

Students' contributions to OSS projects could be introduced during different phases of an undergraduate curriculum. As reflected in Brito et al.'s [1] updated version of the mapping study, some authors decided to introduce OSS on specific labs or seminars during an academic year while other authors prefer to reserve an entire subject to OSS contributions, but there is a lack of previous research about contributing to OSS as the main aim of an Final Degree Project. This is surprising because, in our opinion, FDPs are the most effective way to put into practice this kind of OSS integration in the SE classroom for several reasons, as described in the next section.

Final course work:

We think that in order to allow our students to be involved with such scenarios, they need to have a good understanding of the prerequisites needed to undertake a contribution (solid knowledge about computer programming, data structures and algorithms, software engineering and depending on the project, some other specific knowledge about networking, databases, project management, etc.) This is the case for final-year students developing their FDPs.

Fewer time restrictions: Usually, students of final-year grade have already passed nearly all, if not all, the other courses of their degree, having the final project as the single point of work for their second half of their final year. This allows them to focus on the work during a semester instead of having to manage this alongside other tasks and exercises while doing it. The number of weeks in a semester (15–16 weeks) is roughly the same amount of time allocated by students working on the Google Summer of Code (GSoC) initiative. This Google internship program finances students to write code for open source projects from May to August. Although the economical rewards are of consideration, Silva et al. [6] reported that, typically, the participation of students is motivated by their desire to seeking work experience and build a career portfolio based on their contributions to open source projects.

Initiatives such as the GSoC could be leveraged in many ways. First, they help instructors to choose an interesting project and specific features to contribute to. Second, projects that participate in GSoC always assign a mentor to guide students' efforts (thus, freeing or alleviating the university instructor workload). Finally, students that decide to develop their FDP under the GSoC umbrella can also earn some economical rewards.

External committee-based evaluation: Final projects are evaluated by a board composed by members not linked to the students or even to the supervisor of the project. This allows us to recreate a real-world scenario with a bias-free evaluation process.

A boost in confidence: Students achieving good results on their FLOSS-related final projects get a boost on their confidence just before their immediate immersion in the job market. This confidence is backed up by evidence (for example, in the Github repository of the project or the official documentation) reflected on their CV and portfolio.

Lower student-to-instructor ratio: The FDPs have to be supervised by a lecturer. Usually, each supervisor has one to three students under their direction so they can attend and support them more thoroughly than in a regular (non-final-project) class.

Integrating OSS in the SE curriculum is not without its problems. Instructors will face challenges selecting the project and the specific tasks to conduct, engaging and supporting students, assessing the outcomes or planning how to fit SE educational objectives within a limited time interval.

With the aim of exploring new ways of supporting the teaching of software engineering based on contributions to OSS projects, as well as looking for an alternative to its use in class (where time restrictions and prior knowledge condition its use), this work documents an experience report framed within four final degree projects (FDP) where four undergraduate students with degrees in computer engineering contributed features to a couple of OSS projects.

The next section reviews on how OSS projects have been used in SEE, focusing on two aspects: the software engineering area and the educational purposes covered. The following section details the context in which our students' works were developed and how the projects they contributed to were selected. Next, we describe the challenges faced and how they were addressed, detailing some recommendations for future practitioners interested in repeating the experience. The last section summarizes the lessons learned and suggests future work.

2. Related Work

Our interest is focused on how to leverage OSS in SE education and more specifically, how and why contributing to an OSS in an FDP could benefit the software engineering students.

2.1. OSS in SE

Researchers have addressed many knowledge areas of software engineering education through the use of open source software. Analyzing the last ten years activity, and following an adapted classification by [1] from the original SWEBOKv3 [7], we found some previous experiences tackling software engineering in general, requirements, models and methods, architecture and design, quality, testing, evolution and maintenance, development and construction, process, management and configuration management areas.

Silva et al. [8] report on an experience teaching object-oriented modeling with Unified Modeling Language (UML) diagrams using open source projects. In a postal survey administered to students, they stated enhancement of their ability to interact with real projects and their understanding of source code written by other persons, as well as an improvement in their confidence to face the job market and their communication skills.

Gallagher et al. [9] and Gokhale et al. [10] researched the integration of OSS into an introductory SE course, analyzing software maintenance and evolution. This experience allowed students to face problems associated with software evolution at different difficulty levels, improve their program comprehension, maintenance and reverse engineering skills, by studying projects that usually lacked proper documentation.

Van Deursen et al. [11] promote a collaborative approach to learn about software architecture and design decisions based on open source projects' architecture. They tackled three main challenges: how to teach the theories of software architecture, which are abstract and, therefore, hard concepts for students to master; how to address problems derived from software architecture decisions when they are only visible at scale, and disappear once small example systems are used; and how to teach a proper balance of technical and social skills, something every software architect needs.

Kubincová and Homola [12] analysed the requirements for delivering code review activities (a quality assurance activity) in computer science educational settings. They compared how to perform a code review both in GitLab and a peer-reviewed activity in their Learning Management System.

Tan et al. [13] propose GitHub-OSS Fixit, a course project where Software Engineering students are taught to contribute to open source Java projects by fixing bugs reported in GitHub. Their students reported that this activity helped them to gain confidence on applying the knowledge acquired to real-world scenarios. Similarly, Villarrubia and Kim [14] built an in-house version control system using GitLab to teach students how to collaborate on OSS development projects and instructors to track their students' activities. They report on the added benefits of using the system as a knowledge base repository for future student projects where code and project artefacts can be analyzed and reused.

Tafliovich et al. [15] present the design and delivery of a senior Software Engineering course structured around a collaboration with a large, active FLOSS project. In their work they highlight some aspects that can be effectively taught by promoting students' contributions to open source software, such as soft skills and technical writing, reverse engineering, software modeling, and learning to estimate, plan, and manage new requirements.

Dorodchi and Dehbozorgi [16] proposed eight open source labs for teaching and practicing SE general activities based on FLOSS. They taught how to set-up the development environment; how to communicate inside an OSS community; how to use a source code versioning system, modelling OSS, and requirement analysis (they asked the students to analyse how to make some modifications to a large existing codebase); implementing and testing the proposed new features.

Finally, Deng et al. [17] resort to FLOSS in a quest for providing real-world software quality and testing environment for teaching software engineering, describing five learning interventions that use FLOSS in an undergraduate software testing course. They report an increase in students' confidence in their technical knowledge and skills. Krutz et al. [18] also encouraged their students to choose a real-world, open source project to test throughout an academic term. They reported a significant increase in student enthusiasm in software testing, largely due to being able to test contemporary real-world projects.

FLOSS in Final Degree Projects

This work is more closely related to Stroulia et al. [19] and their Undergraduate Capstone Open Source Project (UCOSP), a course where undergraduates from different institutions across Canada and the United States work together on a real-world FLOSS project with oversight by a mentor, either a faculty or project member. Participants reported benefits such as gaining professional experience, and professional networking. This work differs from Stroulia et al.'s regarding the distribution of the instructors. We report on final degree projects supervised by a single instructor with few students (4) working with an external community of real developers while in the UCOSP experience, groups of nearly 20 students working on their capstone projects are distributed and oversighted by different instructors.

Another close related work is HFOSS (Humanitarian Free and Open Source Software) [20], where undergraduates of computing degree programs are involved in humanitarian software projects development ranging from disaster management to microfinance to election-monitoring applications. Results show students' increased interest in computing and gaining experience in developing distributed software as well as skills in regard to communication and distributed teamwork. HFOSS does not specifically target final degree or capstone projects but their initiative can be used under different academic scenarios: a single assignment, a project deliverable, a capstone or final project or as a topic in a course.

3. Context

Every year at the School of Engineering of Bilbao 4th year undergraduate students in Computer Engineering must develop their Final Degree Project (FDP). Often the project consists of implementing some type of application, usually from scratch, something that does not match what they will find in the labor market [21–23]. Students generally complete good FDP projects but the vast majority end up in the drawers of the library or on the online repository.

One way to motivate computer science students to develop and enhance their software engineering skills is to involve them in the development and maintenance of consolidated open source software projects [24,25]. During two consecutive academic years (2019/2020, 2020/2021) we decided to test this strategy within the context of four final degree projects. Three students (19/20) were assigned to work on the same OSS project (GanttProject (https://github.com/bardsoftware/ganttproject (accessed on 14 May 2021))) while the fourth one (20/21) decided by himself to go solo and contribute to a different OSS project (Ampache, not to confuse with Apache (https://github.com/ampache/ampache (accessed on 14 May 2021)))

As detailed in the Related Work section, there has been quite an interest in academic research works about how to integrate FLOSS activities into the SE classroom. These works have been classified by the systematic mapping study from Nascimento et al. [5] and the updated version by Brito et al. [1] following some main concerns:

- Software Engineering area, with possible values already detailed in the previous section (requirements, models and methods, design and architecture, quality, testing, evolution and maintenance. . .)

- Control Level, i.e., how much control instructors have over the OSS project. Possible values include Full Control (i.e., project core development has been sustained by faculty/staff), Inside Control (i.e., instructors branch the FLOSS code, prepare assignments, and evaluate themselves the students' contribution), and NoControl (i.e., third-party OSS projects).
- Project Choice, i.e., how the OSS projects are chosen. Alternatives include Predefined (i.e., instructors decide the project students work.); Choice List (i.e., students' choices based on a preset list of OSS), and Free Choice (i.e., students seek and decide which OSS they will work with)
- Assessment Type, i.e., the instrument used to assess student learning. Possible values include exams, reports, software artefacts, interviews, surveys, oral presentations, etc.
- Curriculum Choice, i.e., how the content is embodied in the curriculum. Options include Extra Activity, where students worked with OSS in extra activities (e.g., internships); Capstone Project, where students worked with OSS in a capstone project; and Course Working where OSS was a student's assignment in a regular course.

This permits to characterize OSS-in-the-classroom experiences along the coordinates ControlLevel, ProjectChoice, AssessmentType and CurriculumChoice. Our experience resides on the area of software engineering in general (due to being projects that follow the whole software life cycle), free-choice, no-control (students selected by themselves external consolidated OSS projects for their contributions), assessed by a final-degree-project external jury based on a report and an oral presentation.

4. Methods

This section describes how the students were selected and, in turn, how the students selected their projects to contribute to and how instructors and students decided which task to work on in each project.

4.1. Selecting the Students

The aim was to convince some students from course 2019/2020 to focus their final degree project towards improving an open source application used internationally. The instructor (the author of this article) had a list of nearly 20 candidates, students that were in their final year, wanted to develop a final degree project and whose academic background was known to the instructor. Based on their academic curriculum, we filtered an initial list of candidates.

Criteria for Students Selection

Because it was the first year of the experience, we wanted to select a low, manageable number of students so we could be able to carefully supervise all of them. Thus, in order to choose only three of them to take part in the study during the 19/20 academic year, we applied a further set of criteria.

Previous knowledge: The students needed to have completed (and passed with above the average grades) previous subjects about Software Engineering, Data Structures and Algorithms and Computer Programming. Third-party code comprehension, feature location, refactoring, program maintenance, version control and programming are a must for contributing to this kind of mature OSS projects.

Required skills: The students have to demonstrate a good command of the English language. All the conversations around a consolidated OSS project are in English. The students involved need to have, at least, good reading and writing skills for interacting in forums, code reviewing activities, issue management and/or mailing lists. Other desirable skills include self-regulated learning skills to acquire sufficient knowledge of the involved project without needing strong supervision (not only about technical issues but also about the community and project management).

4.2. Initial Training

We scheduled four seminars with the selected students for training them on basic knowledge about OSS development. This kind of training could be used not only for introducing selected students into the realm but also, in future iterations, for helping students interested to gain the required knowledge to participate in this kind of initiative. Specifically, we scheduled the following introductory seminars.

Seminar 1. The open source development model: Describing how an open source project manages a feature cycle (how to manage and understand feature requests, architecture and design discussion, coding, patching, testing and integration, deployment and maintenance).

Seminar 2. Version control management: How to work with Git and GitHub. Basic knowledge about version control concepts (commits, push, pull, branches, forks, pull requests, merge, rebase, resolving conflicts, history management. . .) and branching models (specifically the GitHub Flow)

Seminar 3. Issue management: Centering on GitHub's issue management system, this seminar focuses on how to identify and report a bug in a project. Specifically, how to write a good issue, label it correctly, identify a good beginner-friendly issue, and how to interact efficiently with other developers through issue comments.

Seminar 4. Debugging and feature location: For being able to navigate the source code and locate the root cause of a bug, students need to have a solid understanding of how to use the debugger. Here, we work on understanding the call stack, data inspection, how to navigate and trace code with the debugger and managing exceptions. This seminar includes some exercises to learn basic feature location techniques [26]. For example, departing from a GUI component, we learn how to locate the source code that implements its appearance and behaviour.

After the student selection and initial training, the next task was choosing a good project to contribute to.

4.3. Selecting the Right Project

We can talk at length about what constitutes a good or right OSS project to use in the area of software engineering education [27,28]. In our case, the project had to meet several requirements: be a live, mature (older than 5 years), active project (have commits, meaning code contributions, distributed throughout the year), be developed in Java (the language that students worked in most during their undergraduate courses), not be a trivial project (should contain a considerable number of Java classes) and, if possible, be a project known to the students so they can be engaged more easily.

GanttProject fits all the requirements. It is a Java desktop application comprising almost 800 Java classes and 95,000 lines of code and has been used as a tool in previous subjects for building Gantt charts. Its first commit on GitHub is from June 2010.

Recommendation 1: Based on our experience, in order to smooth students' onboarding, the instructors should involve themselves into the FLOSS project before recommending them to their students. They should dive into the code, trying to understand the basic architecture. Fixing an issue, creating a pull request (PR) and meeting the main developers could be also interesting tasks to be done. Gaining inner knowledge about the project will help to smooth out the entry for new programmers.

During the course 2019/2020, three students of the final year of the computing science engineering degree, O., A., and U. worked hard on selecting, analyzing, developing and fixing bugs from the GanttProject bug list. As explained in the following section, the author also worked with them as an instructor and facilitator for 6 months.

At the beginning of the course 2020/2021, the author received a new proposal from a student very interested in working with Ampache—an open source web based audio/video streaming application developed in PHP since December 2013—to develop new features for the project as part of its final degree project. Having a student going solo and contributing to an open source project was an interesting idea to contrast with our previous experience (with three students involved in the same project), so we accepted the proposal.

Students with no experience contributing to a FLOSS project are not confident enough to choose their initial task and they need support from the community or the instructors [29]. Wolff-Marting et al. [30] propose that open source projects aiming to attract new contributors should label some easy-tasks in their issue management system with tags such as "easy-fix", "beginner-friendly", "easyhack". This could ease the difficulty of the initial work of new contributors.

For selecting the initial tasks, the students and their instructor decided to adopt a mixed strategy. The Issues tab of GanttProject's GitHub repository had (and still has) more than 300 open issues. Some of them are not just software bugs or errors waiting for a fix, but enhancement proposals that need more engineering work. Using the tag filtering feature of GitHub we found that, in this case, no "easy-fix" or similar tags were used for labelling newcomer-friendly issues. Thus, the students were asked to look over the issues list and pre-select a set of three to five potential easy-fix issues. After further reviewing each list of issue candidates, the instructor met with the students and suggested a close list of three issues that students should fix. This process was repeated successively during the course, increasing the difficulty of selected issues, until each student obtained at least three tasks each for working on their projects.

Recommendation 2: Instructors should help students to analyze the project's list of open issues in order to locate a small collection of "feasible" errors and enhancements that can be implemented during the final degree project. They can pre-select some candidate-issues but they should ask students to further analyze the whole list and create a set of 4 or 5 each, whose suitability should be discussed in group.

Recommendation 3: Ideally, students should initially get involved with an easy-to-solve issue, such as eliminating dead code. Then, they should generate a Pull Request with their patch, so that they learn the basis of GitHub Workflow, get their first PR quickly accepted and increase their self-confidence gradually.

5. Results

Whenever a developer wants to contribute to a project in GitHub they need to package their code in a so-called Pull Request and send it to be approved by a senior developer. The approval depends on a code review performed by other developers from the community. After the code review, the PR could be accepted and merged into the main branch or rejected with comments about how to improve their code. As with peer-reviewed articles, usually there are some iterations before a PR is finally accepted.

The students involved in GanttProject (GP) achieved to contribute three pull requests each into the master branch of the repository. Not all of them were accepted. Table 1 summarizes the information about the list of contributed Pull Requests to GP and Table 2 lists the PRs contributed to the Ampache project.

Table 1. Students' contributions to GanttProject. Base URL: https://github.com/bardsoftware/ (accessed on 14 May 2021).

Student	Issue	Contribution URL Path
S1	Sort resources by name	ganttproject/pull/1740
S1	Code Refactor	ganttproject/pull/1730
S1	Option to change cost display format	issues/1659 (not merged)
S2	Add keyboard shortcuts	ganttproject/pull/1697
S2	Remove unused code and methods	ganttproject/pull/1742
S2	Change logo without restart	pull/1723 (not merged)
S3	Code Refactor	pull/1731
S3	Export to CSV tests	pull/1743
S3	Remember Last Import Folder	pull/1678

Table 2. Students' contributions to Ampache. Base URL: https://github.com/ampache/ (accessed on 14 May 2021).

Student	Issue	Contribution URL Path
S4	Customizable login page background	ampache/pull/2647
S4	Added favorite feature for songs	ampache/pull/2700
S4	Playlist duration added to UI	ampache/pull/2614

How to Assess Students' Works

All the students successfully defended their FDPs with outstanding grades. For their assessment we used two different means.

First, we applied the evaluation rubric for FDPs suggested by our university (Publicly accessible FDP evaluation rubric: https://labur.eus/FhXWM (accessed on 14 May 2021)), where the following aspects are taken into account: objective fulfillment, technical complexity, student engagement, general quality of results, report and presentation (formal aspects, writing quality, structure and content, quality of oral presentation and student's answers to the panel members).

Next, in order to discern between outstanding and very good or well done projects, we also followed an ad hoc process. As suggested by Pinto et al. [31], we wanted to determine the nature of the contributions made, classifying it into four classes: forward Engineering (adding new features), reengineering (refactoring), corrective work (fixing bugs) and management work (e.g., updating documentation). Note that the order of the items does not directly reflect complexity (e.g., there could be refactoring-related contributions that might be much more difficult to develop than a simple new feature), but it serves us as an initial guide to assess the difficulty of the work carried out.

Then, for each student, we compute the number of: commits performed, pull requests (proposed, accepted, rejected), documentation pages (web, wiki, readmes, UML diagrams), reported issues, interactions in the developer/help forum, interactions in the mailing list and interactions in the bug management system.

Usually, these numbers reflect the amount of work performed by each student but there are other factors that help grading them. For instance, some PRs are more difficult to get accepted by the OSS project (due to the complexity of the feature that it addresses or because of the code dependencies linked) so they generate much more comments and discussion threads in the code review system (thus, they should be graded -rewarded-accordingly).

So, for each PR we need to find the number of files and lines affected (added, removed or changed) and the number of interactions that each PR generated in the code review process.

Many of the aforementioned numbers can be automatically retrieved using GitHub's API or web scraping scripts but, as reported by Csaba-Zoltán [32], assessing students' projects is usually a subjective task that requires a manual work and a one-by-one analysis.

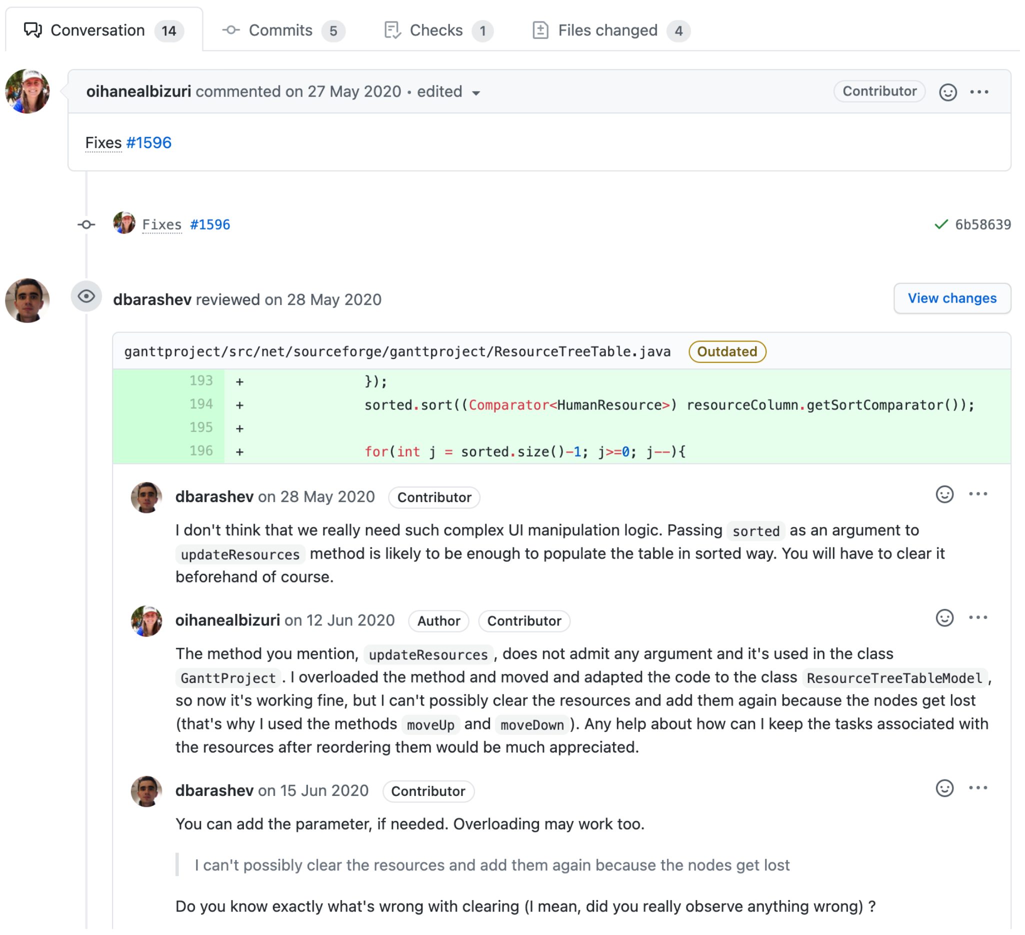
6. Discussion

Working to improve open source applications through final degree projects has brought us multiple benefits and challenges, both for the students, the instructors and for the project itself. The next subsections dive into more details.

6.1. Benefits for the Students

Learn how to work with a branching workflow in Github: GitHub has been extensively used as a collaborative platform for education [33] and specifically software engineering courses [34] . One of the most used features for collaborative development is the GitHub Flow workflow (https://guides.github.com/introduction/flow/ (accessed on 14 May 2021)), a lightweight, branch-based workflow that helps developers to effectively manage code contributions from distributed teams and workgroups. Our students were introduced to this workflow so they could understand how to create a branch of the repository they wanted to contribute to, how to keep their branches synchronized, how to create new feature branches and how to generate Pull Requests and ask community members for reviews. Surprisingly, even though they had developed software projects collaboratively before in other courses, students stated that they had never implemented this workflow until their final project.

Learn to defend and discuss ideas and technical artefacts: Each time students propose a contribution in the form of a PR they learn to cope with the review of their code by external programmers (see Figure 1), usually project lead developers. This type of informal code review, alongside with automation and bug detection tools, simplifies the review process. Many times, even though a student's code works correctly, the lead developers find elements for improvement and suggest refactorings to make the code more readable, maintainable and reusable (non-functional requirements that in many cases go unnoticed by the students during their undergraduate courses).



oihanealbizuri commented on 27 May 2020 • edited Contributor

Fixes #1596

Fixes #1596 ✓ 6b58639

dbarashev reviewed on 28 May 2020 View changes

ganttproject/src/net/sourceforge/ganttproject/ResourceTreeTable.java Outdated

```
+            });
+            sorted.sort((Comparator<HumanResource>) resourceColumn.getSortComparator());
+
+            for(int j = sorted.size()-1; j>=0; j--){
```

dbarashev on 28 May 2020 Contributor

I don't think that we really need such complex UI manipulation logic. Passing `sorted` as an argument to `updateResources` method is likely to be enough to populate the table in sorted way. You will have to clear it beforehand of course.

oihanealbizuri on 12 Jun 2020 Author Contributor

The method you mention, `updateResources`, does not admit any argument and it's used in the class `GanttProject`. I overloaded the method and moved and adapted the code to the class `ResourceTreeTableModel`, so now it's working fine, but I can't possibly clear the resources and add them again because the nodes get lost (that's why I used the methods `moveUp` and `moveDown`). Any help about how can I keep the tasks associated with the resources after reordering them would be much appreciated.

dbarashev on 15 Jun 2020 Contributor

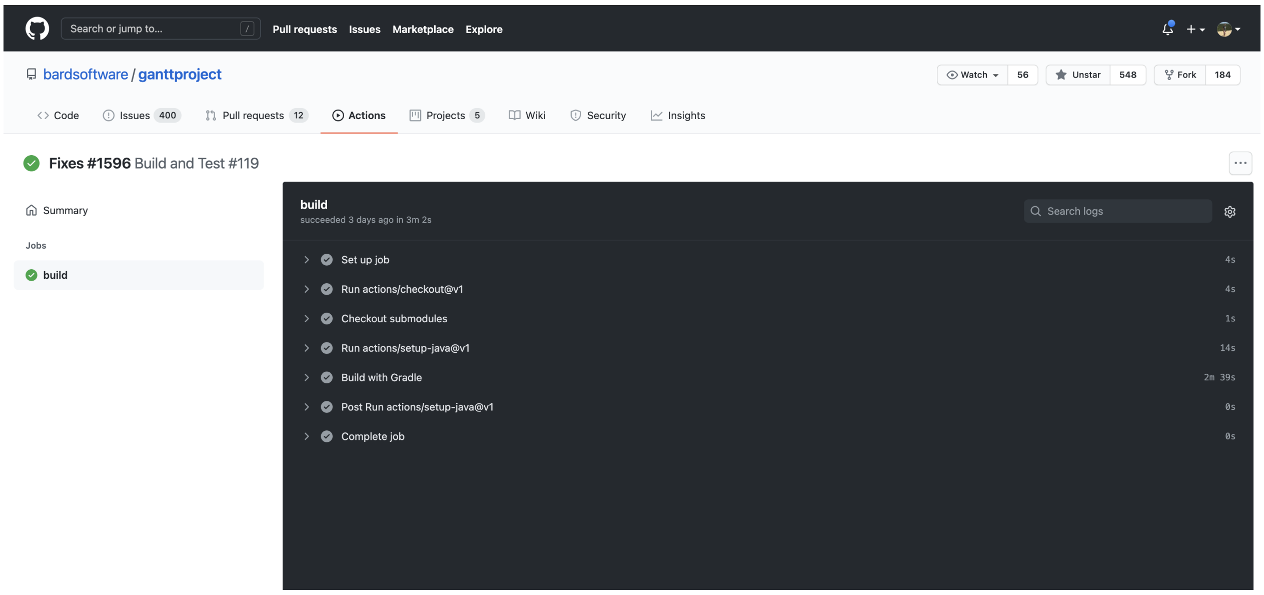
You can add the parameter, if needed. Overloading may work too.

> I can't possibly clear the resources and add them again because the nodes get lost

Do you know exactly what's wrong with clearing (I mean, did you really observe anything wrong) ?

Figure 1. The quality control of consolidated open source projects includes a review of the code contributions. For the students, this was the first time that an external developer reviewed their source code in detail.

Learn to design, implement and pass automatic testing processes: GanttProject includes automated testing features (see Figure 2) as part of their continuous integration process. Students need to understand how tests work and learn how to use them on their own code before submitting their Pull Requests.


Search or jump to... Pull requests Issues Marketplace Explore

bardsoftware / ganttproject Watch 56 Unstar 548 Fork 184

Code Issues 400 Pull requests 12 Actions Projects 5 Wiki Security Insights

Fixes #1596 Build and Test #119

Summary

Jobs

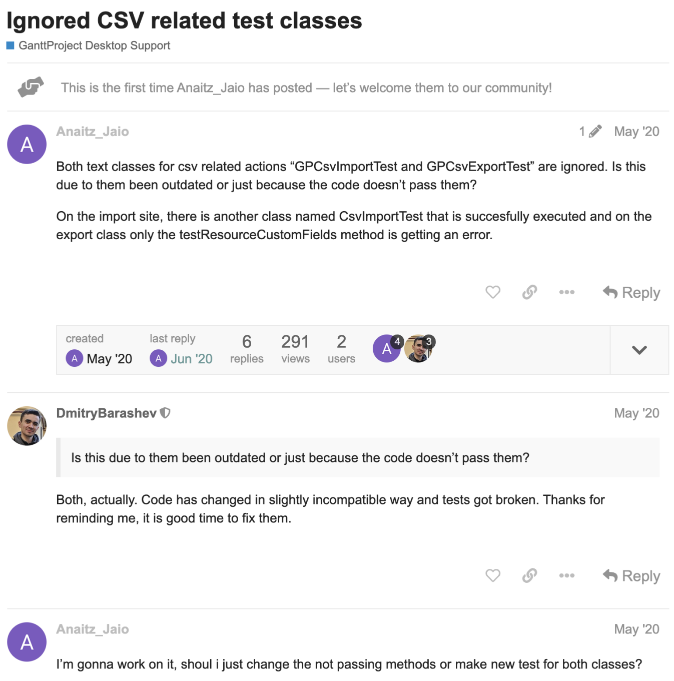
build

build
succeeded 3 days ago in 3m 2s

```
Set up job                          4s
Run actions/checkout@v1             4s
Checkout submodules                 1s
Run actions/setup-java@v1          14s
Build with Gradle              2m 39s
Post Run actions/setup-java@v1      0s
Complete job                        0s
```

Figure 2. The tests include the verification of functionalities on different platforms (Windows, macOS, Linux), something that is not always tested in academic environments.

Practice communication skills: Communicating in English and proposing corrections or designs for new functionalities in the platform's support forum has been another novel task for the involved students (see Figure 3).


Ignored CSV related test classes
GanttProject Desktop Support

This is the first time Anaitz_Jaio has posted — let's welcome them to our community!

Anaitz_Jaio 1 May '20

Both text classes for csv related actions "GPCsvImportTest and GPCsvExportTest" are ignored. Is this due to them been outdated or just because the code doesn't pass them?

On the import site, there is another class named CsvImportTest that is succesfully executed and on the export class only the testResourceCustomFields method is getting an error.

Reply

created May '20 last reply Jun '20 6 replies 291 views 2 users

DmitryBarashev May '20

> Is this due to them been outdated or just because the code doesn't pass them?

Both, actually. Code has changed in slightly incompatible way and tests got broken. Thanks for reminding me, it is good time to fix them.

Reply

Anaitz_Jaio May '20

I'm gonna work on it, shoul i just change the not passing methods or make new test for both classes?

Figure 3. Before solving a problem or beginning an implementation, students should comment on their intentions in the forum.

6.2. Benefits for the Instructors

Even though the instructors are supposed to be involved initially with the open source project in order to provide more effective support for their students (a task that demands quite an effort) this time-investment pays off during the project lifecycle. Indeed, instructors will benefit from the project's community work. These benefits take different forms: the students' works will be carefully reviewed by community members and they may offer continuous feedback through reviewing comments, discussion forums, mailing lists or chat channels, encouraging the students during the whole project and freeing the instructor of those tasks (or sharing the workload involved).

Moreover, the students' interactions and developed projects' artefacts might serve as a collection of real-life examples for future classes. The instructor will be able to draw on those to showcase an up-to-date set of exercises and software engineering best practices based on real cases. The problems and challenges faced by the students involved can also reveal themes and patterns to be fixed- or intensify- in current class syllabus.

6.3. Benefits for the OSS Projects

Finally, GanttProject and Ampache project themselves have also benefited from the experience, specially regarding two aspects:

Getting new features or bug fixes: As seen in Tables 1 and 2, the great majority of the PRs contributed by our students were accepted and merged. These contributions not only enhance the projects at hand but also serve to enrich students' curriculum. Potential job recruiters can verify publicly that our students had really contributed to a public internationally-used project.

Getting reusable documentation: One of the students' goals was to create documentation that will really serve future students and practitioners. Specifically, it was aimed so that other developers can begin contributing with a smoother learning curve than our students had.

Among the generated documentation, the students contributed to the project UML diagrams (class, sequence, communication. . .) and other artefacts such as the kind of diagram from Figure 4 that links names of Java classes with their corresponding component in the GUI. We think that this kind of diagram, collaboratively built by our students, could be very useful for future newcomers.

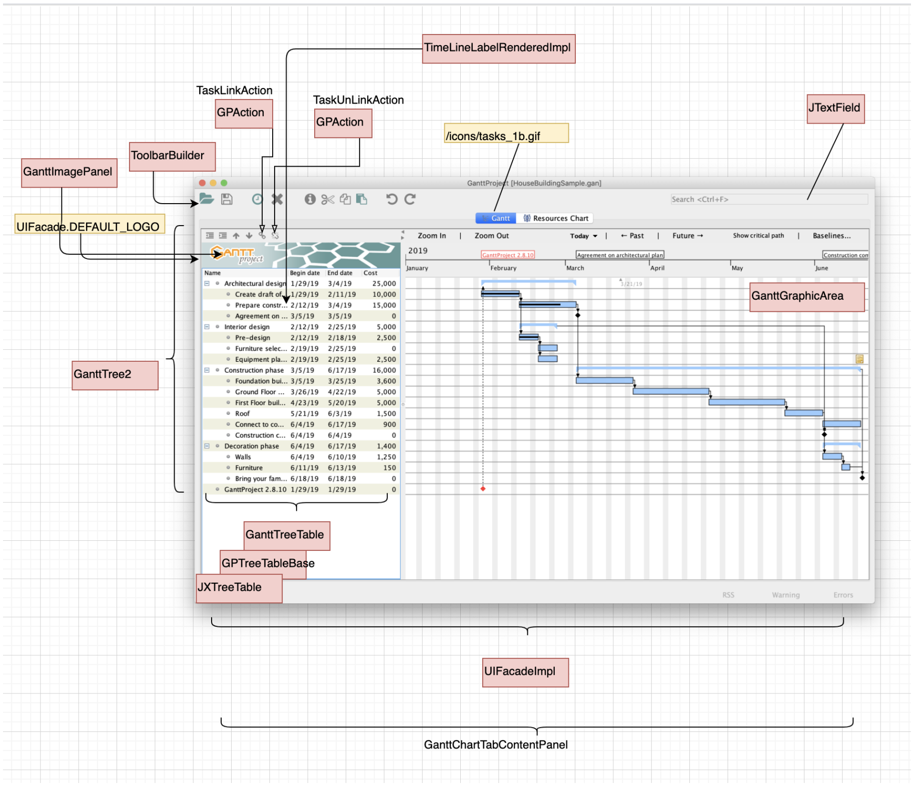


Figure 4. This type of diagram with screenshots of the application labeled with the name of the main classes that implement some of the components can be of great help for new contributors.

6.4. Students' Recommendations

At the end of the projects, the students filled out a survey that asked for recommendations for future interested practitioners. Specifically, they were asked to answer the following questions: (1) What were the most challenging aspects of the project you have worked on? (2) What recommendations would you give to next year's students who want to carry out their final degree projects improving GanttProject/Ampache or any other OSS project? (3) Which aspects of the project were most beneficial to your learning?

A. Most challenging aspects of the project

> "The most complex of all, in my opinion, has been finding the exact point where the error was located or the place where the improvement should be developed"

> "Setting up the development environment, thus being able to compile the application."

> "Understanding the project structure (and code) and breaking the initial fear barriers to collaborate"

> "As students, we are used to starting projects from scratch. However, contributing to an OSS has been a completely different experience. Familiarizing with a project that has more than half a million lines of code is not exactly easy. Especially, taking into consideration that the documentation is usually very shallow or non-existent. The lack of diagrams made it hard even to know where to start working. [. . .] If I had eight hours to contribute to an Open Source Software project, I'd spend six learning the codebase."

Recommendation 4: Navigating the source code and understanding the architecture of consolidated OSS projects can be daunting. Instructors might want to set up some classes about code reading and feature location techniques in order to facilitate students' work.

B. Recommendations for future students

> "Perhaps choosing another OSS software that does not have so many classes and allows you to get a more general picture of the project, since there are many aspects of GanttProject that I still do not know despite having worked on it"

> "Understand from the beginning the structure of the application with which you are going to work, prepare the development environment well and carry out well-planned work, without leaving everything for last minute"

> "I think it would be a good idea to start out with a task related to code analysis, in which each student does research on the project by generating or expanding parts of a previously provided class diagram"

> "Getting involved with the community and start tinkering with the software. Do not wait until you have to start the final degree project to start tinkering with the program. Using the software frequently and taking a high-level overview at the repository are good practices before formally starting to work on the project"

C. Aspects of the project that provided the greatest opportunities to learn

> "As I have progressed in development, I have realized that understanding other people's code and locating parts of the code has become easier and easier for me, and I have done it in less time."

> "The use of tools such as Git and sdkman—a tool to easily manage different versions of the Java Development Kit and Gradle-, as well as the use of the debugger to see the application processes and thus understand the function of various methods and classes in a large project."

> "To read the project structure, source code management with git and to use different data structures from those studied in class."

> "It gives great confidence knowing that, with enough time and effort, you can teach yourself new programming skills. Plus, seeing the contributions I have made integrated into the software each time I use it, is a great motivational boost."

Unlike the students from course 2019/20 that worked collaboratively on GanttProject, the fourth student was working alone with the Ampache project during the academic course 2020/21, so we asked them an additional question: "In contrast with other students, you have worked alone on the project. Did you miss something?" In their answer, the student highlights the importance of the project community to not feel alone:

> "I would not say that I have worked alone on the project. While it's true that I have not worked together with other students, other members from the Ampache community have given me guidance with the pull requests. For example, the users who open the issues are interested in getting the feature integrated with the project, so they are helpful in providing information for the initial user requirements capturing phase. Moreover, developers are helpful in providing useful guidance and making minor fixes to the proposed pull requests"

Recommendation 5: Instructors should create a group on an instant messaging application with the students involved in the initiative. They will help each other, a sense of community will be created and students will feel supported through their projects.

7. Limitations

The main limitation of this study is the low number of involved students. Two main reasons justify this issue: (1) the number of students available for orienting their FDP towards OSS contributions is scarce and (2) the instructor needs to involve him/herself in the process, investing a considerable amount of time guiding and supervising the students, thus limiting the number of potential students under their supervision.

Certainly, this could be seen as a drawback for generalization but this study aims to encourage other instructors and students to follow this path and increase the research works in this area, offering a set of recommendations and detailing the benefits of the investment.

Some instructors are concerned about the validity of this OSS approach. Their argument is that students should use their FDP to gain some practice developing a project from scratch. They argue that students should also demonstrate their ability to select the right technology for solving software engineering problems. These concerns are valid to some extent but it should be noted that first, not all the issues solved in an OSS project are just related to bug fixing (corrective maintenance issues). Some of them are, in fact, full-fledged functional requirements that need to be tackled following all the phases involved in the software development life cycle (requirements gathering, analysis and design, building, testing and deploying). In fact, instructors need to achieve a balance while helping to select a task for their students, offering an easy-enough initial task to solve, that can engage and motivate students, followed by some other more challenging tasks that can serve them to further practice requirements gathering, system analysis and design skills.

8. Conclusions

This work has described a practical experience of four computer science students who have oriented their final degree projects towards the development of improvements to a couple of consolidated OSS projects. For selecting the project, students were supported by the instructor, who helped them to choose the targets to contribute to (GanttProject and Ampache). Each student had to complete the development or fixing of three issues and, following the usual GitHub workflow, aimed to integrate their contributions into the master (main) branch of the selected project. The benefits are multiple, both for students (learning to understand and locate features inside a code base of hundreds or thousands of classes, passing code quality controls, defending design ideas in public groups, working with a distributed version control system), the instructors (freeing them from code reviewing and students' follow-up tasks) as well as for the chosen OSS project itself (get reusable documentation, get fixes or new features). Based on students' work and their responses to a survey, specific recommendations have been proposed, both for instructors, students and future practitioners interested in replicating the process. Among the most complex aspects of contributing to an OSS project during their FDPs students reported the understanding of foreign code structures and locating those parts of the code that affect the error to be corrected or functionality to be implemented. It is therefore important to put more effort and focus on the study of third-party code in the software engineering class, something in which the study of open source software can be of great help. Analyzing how to achieve this integration between contributing to open source projects and the teaching syllabus of a course (such as Software Engineering) is precisely one of the lines of future work. The teaching objectives and competences to be obtained in the course should be linked to possible exercises, tasks and interactions related to the project to which student will contribute, taking into account the time limitations inherent to a four-month course or the different degree of initial knowledge exhibited by the students.

Funding: This research received no external funding.

Informed Consent Statement: Written informed consent was obtained from every subject involved in the study.

Acknowledgments: A short—5 pages—version of the experience presented here was previously published at the SIIE2020 symposium proceedings under the following title: "Capstone Projects Aimed at Contributing to Consolidated Open Source Projects: a Practical Experience". Since then, the manuscript has been fully rewritten and extended (to 14 pages), new data have been added—new students' reports, recommendations and tables—and the experience of a new student that focused their project to contribute to the Ampache project has also been included.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Brito, M.S.; Silva, F.G.; von Flach G. Chavez, C.; Nascimento, D.C.; Bittencourt, R.A. FLOSS in software engineering education: An update of a systematic mapping study. In Proceedings of the XXXII Brazilian Symposium on Software Engineering, SBES '18, Sao Carlos, Brazil, 17 September 2018; Association for Computing Machinery: Sao Carlos, Brazil, 2018; pp. 250–259. [CrossRef]
2. Carrington, D.; Kim, S. Teaching software design with open source software. In Proceedings of the 33rd Annual Frontiers in Education, 2003 (FIE 2003), Westminster, CO, USA, 5–8 November 2003; Volume 3, p. S1C–9, ISSN 0190-5848. [CrossRef]
3. Silva, F.G.; dos Santos, P.E.D.; von Flach G. Chavez, C. Do we use FLOSS in Software Engineering Education? Mapping the Profiles and Practices of Higher Education Teachers from Brazil. In Proceedings of the 34th Brazilian Symposium on Software Engineering, SBES '20, Natal, Brazil, 21 October 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 473–482. [CrossRef]
4. Nascimento, D.M.C.; von Flach Garcia Chavez, C.; Bittencourt, R.A. Does FLOSS in Software Engineering Education Narrow the Theory-Practice Gap? A Study Grounded on Students' Perception. In *Open Source Systems*; IFIP Advances in Information and Communication Technology; Bordeleau, F., Sillitti, A., Meirelles, P., Lenarduzzi, V., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 153–164. [CrossRef]
5. Nascimento, D.M.; Cox, K.; Almeida, T.; Sampaio, W.; Bittencourt, R.A.; Souza, R.; Chavez, C. Using Open Source Projects in software engineering education: A systematic mapping study. In Proceedings of the 2013 IEEE Frontiers in Education Conference (FIE), Oklahoma City, OK, USA, 23–26 October 2013; pp. 1837–1843. [CrossRef]
6. Silva, J.O.; Wiese, I.; German, D.M.; Treude, C.; Gerosa, M.A.; Steinmacher, I. Google summer of code: Student motivations and contributions. *J. Syst. Softw.* **2020**, *162*, 110487. [CrossRef]
7. Bourque, P.; Fairley, R. *SWEBOK v3.0 Guide to the Software Engineering Body of Knowledge*; IEEE Computer Society: Washington, DC, USA, 2014.
8. Silva, F.G.; Brito, M.S.; Tavares, J.V.T.; Chavez, C.v.F.G. FLOSS in Software Engineering Education: Supporting the Instructor in the Quest for Providing Real Experience for Students. In Proceedings of the XXXIII Brazilian Symposium on Software Engineering, SBES, Salvador, Brazil, 23 September 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 234–243. [CrossRef]
9. Gallagher, K.; Fioravanti, M.; Kozaitis, S. Teaching software maintenance. In Proceedings of the 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), Cleveland, OH, USA, 29 September–4 October 2019; pp. 353–362.
10. Gokhale, S.; Smith, T.; McCartney, R. Teaching software maintenance with open source software: Experiences and lessons. In Proceedings of the 2013 IEEE Frontiers in Education Conference (FIE), Oklahoma City, OK, USA, 23–26 October 2013; pp. 1664–1670. ISSN 2377-634X. [CrossRef]

11. Van Deursen, A.; Aniche, M.; Aué, J.; Slag, R.; De Jong, M.; Nederlof, A.; Bouwers, E. A Collaborative Approach to Teaching Software Architecture. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, SIGCSE '17, Seattle, WA, USA, 8 March 2017; ACM: New York, NY, USA, 2017; pp. 591–596. [CrossRef]
12. Kubincová, Z.; Homola, M. Code review in computer science courses: Take one. In Proceedings of the International Conference on Web-Based Learning, Cape Town, South Africa, 20–22 September 2017; Springer: Cham, Switzerland, 2017; pp. 125–135.
13. Tan, S.H.; Hu, C.; Li, Z.; Zhang, X.; Zhou, Y. GitHub-OSS Fixit: Fixing bugs at scale in a Software Engineering Course. *arXiv* **2020**, arXiv:2011.14392.
14. Villarrubia, A.; Kim, H. Building a community system to teach collaborative software development. In Proceedings of the 2015 10th International Conference on Computer Science & Education (ICCSE), Cambridge, UK, 22–24 July 2015; pp. 829–833.
15. Tafliovich, A.; Estrada, F.; Caswell, T. Teaching Software Engineering with Free Open Source Software Development: An Experience Report. In Proceedings of the 52nd Hawaii International Conference on System Sciences, Maui, HI, USA, 8 January 2019.
16. Dorodchi, M.; Dehbozorgi, N. Utilizing open source software in teaching practice-based software engineering courses. In Proceedings of the 2016 IEEE Frontiers in Education Conference (FIE), Erie, PA, USA, 12–15 October 2016; pp. 1–5.
17. Deng, L.; Dehlinger, J.; Chakraborty, S. Teaching Software Testing with Free and Open Source Software. In Proceedings of the 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Porto, Portugal, 24–28 October 2020; pp. 412–418.
18. Krutz, D.E.; Malachowsky, S.A.; Reichlmayr, T. Using a real-world project in a software testing course. In Proceedings of the 45th ACM Technical Symposium on Computer Science Education, Atlanta, GA, USA, 5 March 2014; pp. 49–54.
19. Stroulia, E.; Bauer, K.; Craig, M.; Reid, K.; Wilson, G. Teaching distributed software engineering with ucosp: The undergraduate capstone open source project. In Proceedings of the 2011 Community Building Workshop on Collaborative Teaching of Globally Distributed Software Development, Honolulu, HI, USA, 23 May 2011; pp. 20–25.
20. Ellis, H.J.; Hislop, G.W.; Jackson, S.; Postner, L. Team project experiences in humanitarian free and open source software (HFOSS). *ACM Trans. Comput. Educ. (TOCE)* **2015**, *15*, 1–23. [CrossRef]
21. Chavez, C.; Terceiro, A.; Meirelles, P.; Santos, C., Jr.; Kon, F. Free/Libre/Open Source Software Development in Software Engineering Education: Opportunities and Experiences. *Fórum Educ. Eng. Softw.* **2011**. Avaliable online: https://www.ime.usp.br/~kon/papers/fees11_02.pdf (accessed on 14 May 2021) .
22. Ellis, H.; Morelli, R.A.; Hislop, G.W. Support for educating software engineers through humanitarian open source projects. In Proceedings of the 2008 21st IEEE-CS Conference on Software Engineering Education and Training Workshop, Charleston, SC, USA, 14–17 April 2008; pp. 1–4.
23. Ellis, H.J.C.; Hislop, G.W.; Purcell, M. Project selection for student involvement in humanitarian FOSS. In Proceedings of the 2013 26th International Conference on Software Engineering Education and Training (CSEE&T), San Francisco, CA, USA, 19–21 May 2013; pp. 359–361. [CrossRef]
24. Marmorstein, R. Open source contribution as an effective software engineering class project. In Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education, Darmstadt, Germany, 27 June 2011; pp. 268–272.
25. Müller, M.; Schindler, C.; Slany, W. Engaging students in open source: Establishing foss development at a university. In Proceedings of the 52nd Hawaii International Conference on System Sciences, Maui, HI, USA, 8 January 2019.
26. Dit, B.; Revelle, M.; Gethers, M.; Poshyvanyk, D. Feature location in source code: A taxonomy and survey. *J. Softw. Evol. Process* **2013**, *25*, 53–95. [CrossRef]
27. Pinto, G.H.L.; Figueira Filho, F.; Steinmacher, I.; Gerosa, M.A. Training software engineers using open source software: The professors' perspective. In Proceedings of the 2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T), Savannah, GA, USA, 7–9 November 2017; pp. 117–121. [CrossRef]
28. Smith, T.M.; McCartney, R.; Gokhale, S.S.; Kaczmarczyk, L.C. Selecting Open Source Software Projects to Teach Software Engineering. In Proceedings of the 45th ACM Technical Symposium on Computer Science Education, SIGCSE '14, Atlanta, GA, USA, 5 March 2014; ACM: New York, NY, USA, 2014; pp. 397–402. [CrossRef]
29. Steinmacher, I.; Conte, T.U.; Gerosa, M.A. Understanding and supporting the choice of an appropriate task to start with in open source software communities. In Proceedings of the 2015 48th Hawaii International Conference on System Sciences, Kauai, HI, USA, 5–8 January 2015; pp. 5299–5308.
30. Wolff-Marting, V.; Hannebauer, C.; Gruhn, V. Patterns for tearing down contribution barriers to FLOSS projects. In Proceedings of the 2013 IEEE 12th International Conference on Intelligent Software Methodologies, Tools and Techniques (SoMeT), Budapest, Hungary, 22–24 September 2013; pp. 9–14.
31. Pinto, G.; Ferreira, C.; Souza, C.; Steinmacher, I.; Meirelles, P. Training software engineers using open source software: The students' perspective. In Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training, Montreal, QC, Canada, 25–31 May 2019; pp. 147–157. [CrossRef]
32. Kertész, C.Z. Using GitHub in the classroom—A collaborative learning experience. In Proceedings of the 2015 IEEE 21st International Symposium for Design and Technology in Electronic Packaging (SIITME), Brasov, Romania, 22–25 October 2015; pp. 381–386. [CrossRef]
33. Zagalsky, A.; Feliciano, J.; Storey, M.A.; Zhao, Y.; Wang, W. The emergence of github as a collaborative platform for education. In Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing, Vancouver, BC, Canada, 28 March 2015; pp. 1906–1917.

34. Feliciano, J.; Storey, M.A.; Zagalsky, A. Student experiences using GitHub in software engineering courses: A case study. In Proceedings of the 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), Austin, TX, USA, 14–22 May 2016; pp. 422–431.