Teaching Software Testing with Free and Open Source Software

Lin Deng, Josh Dehlinger and Suranjan Chakraborty
Department of Computer and Information Sciences
Towson University
Towson, MD, USA
{ldeng, jdehlinger, schakraborty}@towson.edu

***Abstract*—Software quality assurance and testing is a vital phase in the software development lifecycle, oftentimes requiring a significant portion of the overall development budget. Providing authentic, real-world software quality assurance and testing experiences within the context of a Computer Science or Software Engineering curriculum is challenging. Based on prior experience leveraging Free and Open Source Software (FOSS) as an instructional medium for teaching Software Engineering concepts, this paper describes five increasingly involved learning interventions that utilize FOSS in a software testing course to provide students real-world hands-on software testing knowledge and experience. The proposed approach will be piloted and assessed in software testing courses at the undergraduate and graduate level.**

***Keywords*-software testing education, free and open source software, test-driven development, continuous integration, software testing automation**

I. Introduction

It is widely acknowledged that software quality is critical to software systems and their users. Low-quality software may lead to significant financial loss or other severe consequences. Therefore, it is estimated that, for most software projects, at least half of the development budget is spent on quality assurance activities, in order to ensure the quality of software [1]. Also, teaching students software testing concepts, techniques and theories is fundamental to most Computer Science undergraduate and graduate programs, so that students, as future industry workforce, understand the importance of software testing.

Teaching and learning software testing are challenging. Real-world software development and testing experience is critical to every Computer Science student. However, sometimes instructors design courses that are overly theoretical, without examples or practical learning activities offering students hands-on experience regarding industry level software testing tools and techniques. Research has indicated that students often do not gain essential skills and knowledge they need to succeed in the IT industry. In particular, there are discrepancies in the fields of software testing, maintenance, and source code control [2]. Most Computer Science programs fail to prepare students for the realities of the IT industry, which leads to students' misunderstanding about software development and testing activities in actual practice [3]. Very often, as instructors of software engineering and testing courses, we are asked by students: what do real-world software defects look like? Students' enthusiasm and eagerness to explore the real IT world can never be overlooked. On the other hand, proprietary software projects from industry usually have restricted access, because of their for-profit nature, which prevents instructors and students from obtaining and utilizing them. Given the length of each academic semester (usually around 15 weeks), the size, the scope, and the number of collaborators of the projects used in testing courses, can never be anywhere close to industry level projects. Lack of knowledge and experience on industry level tools and projects becomes a weakness for many students.

Computer Science education practitioners have started to investigate shifting from greenfield projects, which are developed from scratch, to brownfield projects, for which students join an existing project and participate development and maintenance tasks of this legacy project [4]–[6]. To address the aforementioned challenges, this paper explores the possibility and applicability of using brownfield projects to design a modern software testing course, by proposing an innovative approach that leverages Free and Open Source Software (FOSS) as the major instructional medium of teaching and learning activities in software testing courses to provide students real-world hands-on software testing knowledge and experience. Specifically, this paper presents five learning activities, from rudimentary to comprehensive. Each learning activity includes a set of task directions, required deliverables from students, and an assessment grading rubric for instructors.

The contribution of this paper is that we present five learning activities using FOSS for software testing courses, which provides students real-world industry level hands-on software testing experience and solidifies their understanding of software testing concepts, theories, and practice. These activities expose students to real-world software defects, quality issues, industry technical stacks, and increase their awareness of the importance of software quality, boost their interest in software development and testing, and prepare them for the future workforce. Through these learning activities, students participate in FOSS projects, and become technically confident to software development and testing challenges and tasks. Also, students contribute to FOSS projects and feel the impacts and achievements they make to real-world software projects and their users. By joining FOSS communities, students learn

Authorized licensed use limited to: b-on: UNIVERSIDADE DO PORTO. Downloaded on October 12,2023 at 08:01:56 UTC from IEEE Xplore. Restrictions apply.

teamworking from other senior developers and build their professional network and connections.

This work is part of an initial, larger effort to integrate FOSS activities into the Software Engineering curriculum courses to: 1. provide realistic Software Engineering experiences to apply the concepts, techniques, tools and theories learned in a classroom setting; and, 2. create service-learning opportunities in which students can apply the cutting-edge technology skills in an industry or community project and further develop their Software Engineering skills.

The rest of this paper is organized as follows. Section II introduces the rationale of using FOSS as a pedagogical tool and our experience of using FOSS in Computer Science courses. Section III presents five learning activities using FOSS as the basis. Section IV provides a survey for assessing students' learning outcomes and experience. Section V gives an overview of related research, and the paper concludes and suggests future work in Section VI.

II. The Rationale Behind FOSS as a Pedagogical Tool in Software Testing

Many widely used software applications are FOSS, such as Ubuntu, MySQL, Firefox, etc. There are several advantages of using FOSS as a pedagogical avenue of software testing courses. First, "open source" in FOSS projects indicates that these projects usually are hosted by online repositories, such as GitHub, GitLab, and Bitbucket. Access to these projects is not restricted. Unlike proprietary software, students can easily obtain all the source code of FOSS projects. Second, many FOSS projects have a community of experienced developers who constantly contribute to the projects. It is fairly easy for students to join the community, contact and learn from these senior developers. Research of Wilson et al. found that, by engaging into the FOSS community, students' academic performance can be improved [7]. Third, FOSS projects are usually well-documented with Wiki or other similar platforms and well-maintained by their community using issue tracking systems. Students can easily have a jump start from these systems. Fourth, various size and scope of FOSS projects provide instructors the freedom to tailor their courses depending on the diverse background of their students.

At Towson University, using HFOSS (Humanitarian FOSS) in teaching Computer Science related courses has been practiced, evaluated, and improved upon over the years. Project Wavyleaf [8] was initiated in Fall 2013 and was the first service-learning project undertaken by the Mobile Application Development Lab at Towson University. The Mobile Application Development Lab was established in Spring 2013 as part of a larger NSF-TUES grant [9] to provide a permanent application development/research lab for students working on mobile computing within the department and to develop external, service-learning projects to provide student teams with experience on real-world FOSS projects. Multiple undergraduate students in Computer Science participated the project, between Fall 2013 and Fall 2015. These students participated through an independent study, for-credit course or through a paid research assistant position and managed by one biology faculty member (i.e., the "client") and two Computer Science faculty members (i.e., the "project managers"). The project was managed using GitHub repositories.

TABLE I: Student Experience Survey Results

Survey Question	Results
I can use a software process to develop an HFOSS project.	4.4 / 5
I am sure that I can actively participate in an HFOSS community to develop a software project.	4.6 / 5
I have a greater awareness of the potential for computing to benefit society due to working on an HFOSS project.	4.0 / 5
I wanted to work on an HFOSS project because I want to help the people who would benefit from the software.	4.0 / 5
Knowing that my project will help people motivates me to do my best on the HFOSS project.	4.2 / 5

To evaluate students' experience, students were asked to anonymously and voluntarily complete surveys, based on evaluation surveys introduced by Ellis et al. [10], upon completion of their time with the project to assess their perceived experience in Software Engineering and HFOSS while working on Project Wavyleaf. Table I provides a representative average of the survey results presented on a scale of 1-5 with 1 equating to Strongly Disagree and 5 equating to Strongly Agree.

Throughout the project, students expressed enthusiasm in their participation in FOSS projects for the following reasons:

- Real-world project involving real-world technologies. The primary driver for most students who got involved with Project Wavyleaf was to be able to gain experience in mobile application development in Android or iOS. Several students expressed that technology skills gained and, more importantly, the open source/publicly availability of their work enabled their internship/job search as they were asked to present their application and repository code as a part of interviews.
- Humanitarian/environmental awareness. Several students expressed that working on a project that was "doing good" for the environment motivated them far more than a typical classroom "toy" project or research project because they felt they were giving back to their local community and/or donating some of their time/talents to a good cause.

These positive student outcomes and rich experience of using FOSS projects in Computer Science courses provide us a strong belief of the applicability of FOSS projects in software testing courses.

III. Learning Activities Using FOSS

This section introduces learning activities designed using FOSS to teach software testing.

A. Selection of FOSS Projects

There are more than 44 million open source repositories on GitHub [11]. However, not every open source project is suitable for students to learn software testing. Many projects are not developed, maintained, or managed appropriately. A list of FOSS project requirements must be outlined at

Authorized licensed use limited to: b-on: UNIVERSIDADE DO PORTO. Downloaded on October 12,2023 at 08:01:56 UTC from IEEE Xplore. Restrictions apply.

the beginning. For example, we recommend selecting FOSS projects that are mainly developed in Java, because Java is an object-oriented programming language introduced in our entry-level programming courses. We also suggest selecting FOSS projects that are actively developed and have multiple active committers. If a project is not actively maintained, students will not receive any feedback or help for their work involving the project. Additionally, the size of selected FOSS projects cannot be too small or too large. Huge complicated projects can easily make students lost in digging the structure of the projects, while small projects can downgrade the achievement and contribution of students.

We provide the following three different project selection strategies:

- *Dictator strategy*. Instructors choose one FOSS project for the whole class, so that students do not need to search, evaluate, and understand multiple different projects. Sometimes, given the limited technical background and experience of students, it is likely that students underestimate the complexity of FOSS projects. This also allows the instructor to screen potential projects to ensure that they can be compiled and built, and the development environment can be established; oftentimes, the difficulty of these two initial steps is overlooked and can become a significant hindrance to achieving the course objectives when adopting FOSS.
- *Free-for-all strategy*. Students are free to choose any FOSS projects they would like to work on. Most software testing courses are offered at junior, senior, or graduate level, in which, students' technical background may be diverse within a class. One project may become too easy for some students, while too difficult for others. This strategy provides students the freedom to select a project that is suitable to themselves but does not assess the ease of which it can be compiled and built a priori.
- *Hybrid strategy*. This strategy lets instructors select several candidate FOSS projects for the whole class. Then, students can choose one from the candidates, based on their preference, background, and experience. This strategy prevents students from underestimating the difficulty of projects, but also preserves the freedom of students to choose a suitable one and takes students' diverse background into consideration.

B. Learning Activities

Five major learning activities are designed to utilize FOSS projects to teach software testing, including each one's directions, student deliverables, and recommended assessment criteria. The difficulty levels of these learning activities span from easy to hard. In addition, these activities include necessary software development and testing activities used in most projects in industry, and thus help students prepare for their future job hunting.

1) Fork a project and execute existing JUnit tests: The very first step after selecting an FOSS project is to fork, clone, and get familiar with the project. Executing existing JUnit tests becomes the best method that helps students understand the project.

a) *Directions*: Students need to: i) setup their GitHub accounts and install Git on their local machine; ii) fork the project into their accounts; iii) clone the project to their local machine; iv) import the project into an IDE, such as Eclipse or NetBeans; v) locate JUnit test cases in the project; vi) execute these existing JUnit test cases and obtain test results.
b) *Deliverables*: Students need to submit: i) test results; ii) screenshots of each step and a short discussion of the process, regarding how they follow the directions to finish the assignment, any possible difficulties, and how they overcome the difficulties.
c) *Assessment*: This activity sets the foundation for other activities. Students are expected to accomplish all the requirements. Table II lists the assessment criteria for this activity.

TABLE II: Assessment of Activity 1

Criteria	Level 1 (Poor)	Level 2 (Fair)	Level 3 (Excellent)
The project is cloned and imported into an IDE.	The project cannot be imported into the IDE.	The project is imported, but with some errors.	The project is imported without errors.
The tests included in the project are executed.	The tests cannot be located.	The tests are located but cannot execute with errors.	The tests are executed without problem.

2) Design new JUnit tests for a method to achieve a testing coverage: Testing coverage criteria are rules that impose test requirements on test cases and they are the most important quantifiable quality attribute for test cases. On the other hand, introducing test automation frameworks, such as JUnit, is an indispensable step for every software testing course. This activity helps students understand the concept of testing coverage and practice JUnit testing. Specifically in this activity, we use branch coverage as an example, but the activity applies to any other testing coverage criteria.

a) *Directions*: Students need to: i) identify a method under testing that is not touched by any test; ii) identify all the branches in this method; iii) for each identified branch, design at least one test input and provide its corresponding expected output; iv) implement test inputs and expected outputs with JUnit test automation framework; v) execute these JUnit test cases and obtain test results; vi) commit all newly designed tests to the GitHub repository; vii) submit a pull request to the community of the project.
b) *Deliverables*: Students need to submit: i) JUnit tests source code; ii) test results; iii) screenshots of each step and a short description of each identified branch, its test inputs and expected outputs.
c) *Assessment*: JUnit forms the basis of automated software testing, which is required for almost all industry projects. We expect students to be able to write JUnit tests profi-

Authorized licensed use limited to: b-on: UNIVERSIDADE DO PORTO. Downloaded on October 12,2023 at 08:01:56 UTC from IEEE Xplore. Restrictions apply.

TABLE III: Assessment of Activity 2

Criteria	Level 1 (Poor)	Level 2 (Fair)	Level 3 (Excellent)
Branches in the method are identified.	Cannot identify branches.	Branches are identified but some branches are missing.	All branches are identified.
Test inputs and outputs for each branch are designed.	Expected outputs are not designed. Tests are not designed for each branch.	Missing branches or tests.	Each branch has at least one test with inputs and expected outputs.
Tests are implemented using JUnit.	Tests are not implemented using JUnit.	Not all tests are implemented using JUnit.	All tests are implemented and executed without problems.
Commit and pull request.	Tests are not committed. Pull requests are not submitted.	Errors or problems while committing tests or submitting pull requests.	Tests are committed to students' GitHub repository. Pull requests are submitted to the original FOSS repository.

TABLE IV: Assessment of Activity 3

Criteria	Level 1 (Poor)	Level 2 (Fair)	Level 3 (Excellent)
The bug report has necessary information for reproducing the issue.	The report has no information for reproducing the issue.	Partial information is provided.	The report has detailed steps and screenshots.
The bug report follows a required format.	The report does not follow any format.	A format is followed but some key items are missing.	The report follows a format and includes all necessary items.
The bug report includes actual results and expected results.	No results are included.	Either actual results or expected results are missing.	Both types of results are provided in detail.
Labels of bug reports are used appropriately.	No labels are used.	Some labels are used.	The report uses labels appropriately, such as unconfirmed, confirmed, fixed, closed.
The bug report is assigned.	The report is not assigned.		The report is assigned.

ciently. Also, from this activity, students start to experience, participate and contribute to the FOSS project, and communicate to the community of the project. Depending on different FOSS projects, we do not expect the pull requests from students to be approved or responded. Table III lists the assessment criteria for this activity.

3) Write bug reports and get familiar with issue tracking systems: Whenever an unexpected issue happens during the execution of a software system, a bug report should be filed, so that the developers and maintainers of the system can be aware of and start to fix the issue. Many FOSS projects leverages certain bug tracking systems. Bug reports are critical while fixing problems occurred in software systems. Well-written bug reports can significantly save the time and effort that people spend on fixing problems, while poorly-written reports may waste time and budgets of the project.

Therefore, learning how to write good bug reports and become familiar with issue tracking systems are essential to students. Through this activity, students learn how to use these systems to manage defects and issues during the software development process. It is understandable that identifying a real defect in a well-maintained FOSS project may not be practical to students. As a result, in this learning activity, we recommend instructors manually create defects in the project for students to practice. Also, it is very important that instructors provide students guidelines of good bug reports, so that students can follow these guidelines to practice. In this activity, we use the issue tracking system in GitHub as an example, but the activity applies to any other issue tracking systems.

a) *Directions*: After the instructor creates a software fault in the project, students need to: i) identify the fault created by the instructor; ii) create a bug report in the GitHub issue tracking system following the guidelines introduced by the instructor; iii) assign the issue to him/herself or other team members; iv) label the issue accordingly; v) try to reproduce the issue; vi) fix and close the issue.
b) *Deliverables*: Students need to submit: i) a bug report; ii) fix and update to the defect version created by the instructor.
c) *Assessment*: Students are expected to finish a bug report following the guidelines. Table IV lists the assessment criteria for this activity.

4) Setup Continuous Integration (CI) services and practice regression testing: It is common when a software system is changed, either adding a new feature or fixing a fault, new faults are introduced in to the system. Thus, whenever there is a change in the system, all tests or at least a selected test set should be re-executed to ensure the system is still performing expected behaviors, which is called regression testing. Any software project that values its quality enforces regression testing, so it is essential to let students understand and practice regression testing. Additionally, for FOSS, continuous integration (CI) becomes a common practice of regression testing and ensure high quality. Figure 1 illustrates the process of CI. This activity also sets the foundation of the subsequent learning activity.

a) *Directions*: Students need to: i) setup CI service and environment using Travis-CI (a CI web platform [12]); ii) connect the GitHub repository with Travis-CI; iii) test CI service, by committing a change to the GitHub repository, which leads to a failed CI test; iv) verify a notification is received from Travis-CI; v) commit another change that fixes the failure; vi) verify that a notification is received from Travis-CI that the CI test is successfully finished.
b) *Deliverables*: Students need to submit: a document with screenshots describes each step is completed.
c) *Assessment*: Table V lists the assessment criteria for this activity.

Authorized licensed use limited to: b-on: UNIVERSIDADE DO PORTO. Downloaded on October 12,2023 at 08:01:56 UTC from IEEE Xplore. Restrictions apply.

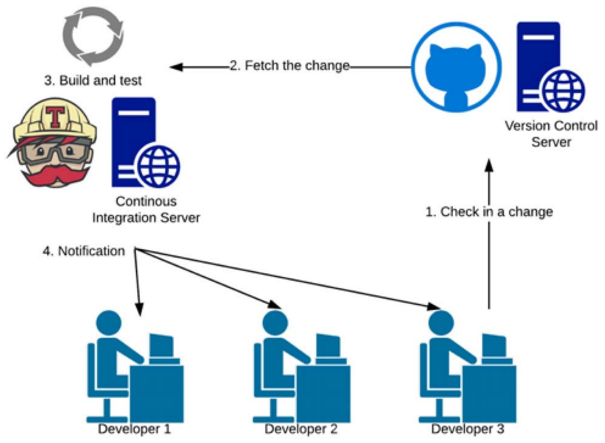

Fig. 1: Continuous Integration Process

TABLE V: Assessment of Activity 4

Criteria	Level 1 (Poor)	Level 2 (Fair)	Level 3 (Excellent)
CI service and environment is set up.	CI setup is not correctly done.	CI works with errors.	CI is working correctly.
Changes are introduced.	No changes are introduced.	Changes are introduced and committed with errors.	Changes are introduced and committed correctly.
Notifications are received.	No notifications.	Notifications are sent, but errors occur.	Notifications are received.

5) Practice Test-Driven Development (TDD): Agile software development process has become a dominating approach in IT industry, by welcoming changes and quickly accommodating customers' feedback and requirements. TDD [13] is an agile approach that emphasize quality improvement in software projects [14]. As software testing is a course that aims to increase students' awareness on software quality, teaching TDD and providing students hands-on experience to practice TDD become a new goal of software testing instructors. As the most complicated learning activity, this learning activity is designed based on this purpose and requires all the technical setups and experience from the prior four learning activities. In this activity, instructors select a set of unimplemented features from the FOSS project, according to students' diverse background.

a) *Directions*: Students need to: i) choose an unimplemented feature provided by the instructor and design a set of new JUnit tests for this new feature; ii) commit these new tests to the GitHub repository and observe Travis-CI executing all the tests; iii) verify that Travis-CI sends a notification that the new tests fail; iv) write new code to get new tests pass; v) commit new code to the GitHub repository and observe Travis-CI executing with new updates; vi) verify that a notification is received from Travis-CI that all tests pass; vii) repeat from step i until all the new features are implemented; viii) initiate a pull request to the original FOSS repository with all the new features.

b) *Deliverables*: Students need to submit: i) JUnit tests source code; ii) test results; iii) source code of new features; iv) screenshots of each step and a short description of how TDD is used in this development.

c) *Assessment*: Table VI lists the assessment criteria for this activity.

TABLE VI: Assessment of Activity 5

Criteria	Level 1 (Poor)	Level 2 (Fair)	Level 3 (Excellent)
New tests are designed to drive the development	New tests are not designed.	Some new tests are designed, but not sufficient.	Each new feature maps to a set of new tests that drive the development.
TDD is followed.	TDD process is not followed.	TDD process is partially followed.	TDD process is strictly followed.
New features are implemented.	New features are not implemented.	New features are partially implemented with errors.	All required new features are fully implemented.
A pull request is initiated.	No pull request has been initiated.		A pull request initiated successfully.

IV. Survey Designed to Evaluate Outcomes

To evaluate students' experience after finishing the five learning activities designed in this paper, a survey has been designed to collect students' opinions and assess students' learning outcomes. Specifically, students are asked to anonymously complete 13 survey questions (listed in Table VII) before and after the course (Students answer each question on a scale of 1-5 with 1 equating to Strongly Disagree and 5 equating to Strongly Agree).

These survey questions help instructors collect feedback regarding whether using FOSS is an ideal avenue in teaching software testing, by assessing how confident students are towards testing of FOSS, how much technical experience they gain, and how deep understanding they obtain regarding software testing, upon completing these activities. Also, the results from these survey questions can lead to reflections of instructors, in order to improve with additional learning activities.

V. Related Work

Leveraging FOSS as an effective avenue in Computer Science education has been piloted, evaluated, and investigated for years. Courses incorporating with FOSS projects range from senior capstone [15], software design [16] and architecture [17], software engineering [18], [19], to CS2/data structure [20], service learning practicum [21], and distributed software engineering [22]. Focusing on software testing concepts, theories, and practice, this paper is among the first ones that design learning activities with a new concentration on software quality.

Authorized licensed use limited to: b-on: UNIVERSIDADE DO PORTO. Downloaded on October 12,2023 at 08:01:56 UTC from IEEE Xplore. Restrictions apply.

TABLE VII: Student Experience Survey

ID	Survey Question
1	I can describe how FOSS projects are tested.
2	I understand the importance of thoroughly testing code.
3	I can test large scale FOSS projects to ensure that it does what it is supposed to do.
4	I always thoroughly test my code.
5	I can list and utilize different testing coverage criteria.
6	I can describe how large scale FOSS projects with multiple programmers work are organized, communicate, and tested.
7	I can use version control tools, such as Git.
8	I can use CI tools, such as Travis-CI.
9	I know how to set up a development environment for a large scale FOSS project.
10	I am comfortable that I could participate in the development and testing of a real-world FOSS software project.
11	I can use TDD to develop an HFOSS project.
12	I am sure that I can actively participate in an HFOSS community to develop and test a software project.
13	I have gained some confidence in collaborating with professionals from a variety of locations and cultures.

Benefits and positive impacts on students and instructors of using FOSS have been investigated for years. Through a six-year study, involving nearly 200 undergraduate students from four academic institutions in the United States, Ellis et al. found that students' motivation to study computing is significantly improved after enrolling courses using FOSS projects and students' learning experience of software development and process is also enhanced [10]. Also, FOSS is considered as very effective at attracting underrepresented minorities in learning computing majors [23].

VI. Conclusions

As one of the most important phases in the software development lifecycle, testing directly impacts the quality of software systems. Teaching software testing concepts, theories, techniques, and practice becomes an essential task for computing education instructors, professors and practitioners. However, teaching software testing is always challenging, considering the rapid development of IT techniques. Students often find it difficult to gain real-world project experience from regular courses.

FOSS is a major type of software programs in the industry. Many software programs people use every day are FOSS. Its inherent characteristics and advantages, such as its friendly and open culture, unrestricted access, and well-maintained wiki and documents, make it an ideal avenue in software testing education. Aiming to solve the challenges in teaching software testing, this paper proposes five learning activities that leverage FOSS projects in education. Their difficulty level ranges from simple to comprehensive. Each learning activity has detailed instructions, deliverables and grading rubrics. A pre/post survey is also provided to instructors to assess and evaluate students' learning outcomes and obtain students' feedback for future improvement. Through using FOSS in software testing education, we expose students to real-world software projects, particularly, real software defects and industry level software testing techniques. Students gain hands-on experience, form their professional network by communicating with senior software engineers around the world, and become confident in their technical knowledge and skills.

For future work, we plan to streamline and utilize these new FOSS learning activities in our software testing courses and collect students' feedback. We aim to improve and design a modern software testing course that can truly help and benefit undergraduate and graduate students, so that more and more students would like to learn software testing and become interested in software testing. Ultimately, well-trained students can join the IT industry and contribute to the high quality of software products.

Acknowledgment

We gratefully acknowledge the support from the Professors' Open Source Software Experience (POSSE) [24]. Deng is supported as a Jess and Mildred Fisher Endowed Professor of Computer and Information Sciences from the Fisher College of Science and Mathematics at Towson University.

References

[1] P. Ammann and J. Offutt, *Introduction to Software Testing*. Cambridge: Cambridge University Press, 2016.
[2] M. Exter, "Comparing educational experiences and on-the-job needs of educational software designers," *SIGCSE 2014 - Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, pp. 355–360, 2014.
[3] L. A. Sudol and C. Jaspan, "Analyzing the strength of undergraduate misconceptions about software engineering," *ICER'10 - Proceedings of the International Computing Education Research Workshop*, pp. 31–39, 2010.
[4] C. D. Hundhausen, N. H. Narayanan, and M. E. Crosby, "Exploring studio-based instructional models for computing education," *SIGCSE'08 - Proceedings of the 39th ACM Technical Symposium on Computer Science Education*, pp. 392–396, 2008.
[5] A. Begel and B. Simon, "Novice software developers, all over again," *ICER'08 - Proceedings of the ACM Workshop on International Computing Education Research*, vol. 1, no. 425, pp. 3–14, 2008.
[6] M. Craig, P. Conrad, D. Lynch, N. Lee, and L. Anthony, "Listening to early career software developers," *Journal of Computing Sciences in Colleges*, vol. 33, no. 4, pp. 138–149, 2018.
[7] D. Wilson, D. Jones, F. Bocell, Joy, C. ee, J. Kim, N. Veilleux, T. Floyd-Smith, R. Bates, M. Plett, D. Wilson, D. Jones, F. Bocell, J. Crawford, M. J. Kim, N. Veilleux, T. Floyd-Smith, R. Bates, and M. Plett, "Belonging and Academic Engagement Among Undergraduate STEM Students: A Multi-institutional Study," *Research in Higher Education*, vol. 56, pp. 750–776, 2015.
[8] V. Beauchamp, J. Dehlinger, and S. Kaza, "Project Wavyleaf: A Towson University Citizen-Science Initiative," in *Proceedings of the 2015 Ecological Society of America Annual Meeting*, 2015.
[9] S. Azadegan, J. Dehlinger, and S. Kaza, "Revitalizing the computer science undergraduate curriculum inside and outside of the classroom using mobile computing platforms (abstract only)," in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 713. [Online]. Available: https://doi.org/10.1145/2538862.2544302
[10] H. J. Ellis, G. W. Hislop, S. Jackson, and L. Postner, "Team project experiences in humanitarian free and open source software (HFOSS)," *ACM Transactions on Computing Education*, vol. 15, no. 4, 12 2015.
[11] "GitHub Octoverse 2017 — Highlights from the last twelve months." [Online]. Available: https://octoverse.github.com/
[12] "Travis CI - Test and Deploy Your Code with Confidence." [Online]. Available: https://travis-ci.org/
[13] K. Beck, *Test-driven development : by example*. Addison-Wesley, 2003.
[14] Y. Rafique and V. B. Mišić, "The effects of test-driven development on external quality and productivity: A meta-analysis," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 835–856, 2013.

Authorized licensed use limited to: b-on: UNIVERSIDADE DO PORTO. Downloaded on October 12,2023 at 08:01:56 UTC from IEEE Xplore. Restrictions apply.

[15] S. Jackson and H. Ellis, “Supporting HFOSS using scrum in a capstone course,” *ACM SIGCAS Computers and Society*, vol. 45, no. 2, pp. 36–37, 2015.

[16] D. Carrington and S. K. Kim, “Teaching software design with open source software,” in *Proceedings - Frontiers in Education Conference, FIE*, vol. 3. Institute of Electrical and Electronics Engineers Inc., 2003, pp. S1C9–S1C14.

[17] C. Costa-Soria and J. Pérez, “Teaching software architectures and aspect-oriented software development using open-source projects,” *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE*, vol. 1241, no. 1, p. 385, 2009.

[18] H. J. Ellis, G. W. Hislop, and R. A. Morelli, “A comparison of software engineering knowledge gained from student participation in humanitarian FOSS projects,” *ITiCSE'11 - Proceedings of the 16th Annual Conference on Innovation and Technology in Computer Science*, p. 360, 2011.

[19] J. Nandigam, V. N. Gudivada, and A. Hamou-Lhadj, “Learning software engineering principles using open source software,” *Proceedings - Frontiers in Education Conference, FIE*, pp. 18–23, 2008.

[20] S. Weiss, J. Klukowska, and D. Burdge, “Injecting Open Source Content Into CS2/Data Structures Courses,” in *The Twenty-Second Annual Consortium For Computing Sciences in Colleges, Northeastern Conference,*, 2017.

[21] A. Bloomfield, M. Sherriff, and K. Williams, “A Service Learning Practicum capstone,” *SIGCSE 2014 - Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, pp. 265–270, 2014.

[22] E. Stroulia, K. Bauer, M. Craig, K. Reid, and G. Wilson, “Teaching distributed software engineering with ucosp: The undergraduate capstone open-source project,” *Proceedings - International Conference on Software Engineering*, pp. 20–25, 2011.

[23] H. Ellis, D. Burdge, R. Morgan, P. Ordóñez, and K. Alkoby, “Using Humanitarian Free and Open Source Software (HFOSS) to Attract the Underrepresented to Computer Science,” in *2015 ACM Richard Tapia Celebration of Diversity in Computing*, Boston, MA, 2015.

[24] “POSSE - Foss2Serve.” [Online]. Available: http://foss2serve.org/index.php/POSSE

Authorized licensed use limited to: b-on: UNIVERSIDADE DO PORTO. Downloaded on October 12,2023 at 08:01:56 UTC from IEEE Xplore. Restrictions apply.