# Utilizing Open Source Software in Teaching Practice-based Software Engineering Courses

Mohsen Dorodchi
Department of Computer Science
University of North Carolina, Charlotte, NC, USA
Mohsen.Dorodchi@uncc.edu

Nasrin Dehbozorgi
Department of Computer Science
University of North Carolina, Charlotte, NC, USA
NDehbozo@uncc.edu

*Abstract*— **Software engineering courses face the challenge of covering all the stages of analysis, development, maintenance, and support while addressing practical issues such as dealing with large codebase. Free and open source software (FOSS) and more specifically humanitarian free and open source software (HFOSS) have been used by many educators to bring many add-ons to computer science education such as innovation and motivation. In addition, FOSS/HFOSS could give a better understanding of real world projects to students. In this work, we are looking at some activities developed for teaching upper division undergraduate and graduate software engineering courses using open source software projects and analyze the impacts of using this approach on students.**

***Keywords— Software Engineering; Open Source;***

## I. INTRODUCTION

Open Source software promotes information sharing which is considered as a powerful and positive notion [3]. An unbiased evluation of common open source software (OSS) products, developed mostly by unpaid volunteers, is often of superior quality compared to the software developed by some of the most well-known companies employing unquestionably extremely intelligent people [5]. Every member of the community contributes to so called "collective open source knowledge" [5] by providing new ideas, designing new features, providing experimental data and deployment logs. Therefore, continuous improvements of OSS by community members result in more reliable and flexible tools with higher quality than commercial counterparts [5]. Consequently, some popular open source software products undergo continuous development and tend to be more reliable than their commercial peers [8]. Consequently, tracking such changes is an outstanding learning tool where a learner could study and review the evolving code continuously to understand its logic and design as well as developer's approaches. The user of the code might also make some modifications to the design by adding new features to improve the tool as well as learning how to work with existing software. Open source has been used for educational purposes based on some pedagogical reasons as listed in the following [3]:

- in Open Source the emphasis is on principles and theories related to the field (e.g. programming languages, etc.)– not specific software tools of a particular software provider;
- it is possible to freely combine learning materials with open source software (e.g. text materials and simulations);
- it brings new possibilities for students' creativity.

Students need to learn the process of studying and understanding open source software, which requires them to learn about prior developers' achievements as well as conducting further discussions and inquiries to result in future enhancements.

Upon learning open source code, users have the freedom to customize the solution for their own use [6]. Martin [6] has also reported that since many OSS projects have hundreds of developers around the world, such softwares keep their pace with technology trends better. With so many contributors to such projects, the productivity is maximized while the need for training and technical understanding of how to interact with them is minimized. Therfore, open source software could help students in learning how to customize existing codes based on given requirements. Consequently, open source encourages students to innovate as well as plan and execute continuous improvements [6].

Use of open source (OS) and free and open source software (FOSS) in education, in general as well as computer science education in particular, have been considered by many educators. For example, authors in [1] discuss about why and how open source is used more and more in undergraduate computer science education. One of these areas is the software development and engineering itself. Most undergraduate students always write their programs (and algorithms) from scratch while working alone, which is in total conflict with software development pattern of industries. In other words, students do not normally work on someone else's code. Therefore, as they enter the job market and are placed in a team of developers working with existing codes, they may face new challenges. On the other hand, the code size in industrial projects is much larger than academic projects as far as the number of lines and the number of resources involved. This mismatch may create additional problems for fresh computer science graduates in workplace. We strongly believe that integration of open source into software engineering courses infuses strong features from professional software engineering world to remedy a straightforward solution to the aforementioned issues.

Educators have been using OSS as an integral part of their teachings for a long time. Some have been using open source just for practice purposes while others have gone beyond classroom experiments. According to [1], computer science undergraduate education can take advantage of open source culture and provide a modern curriculum progression. They believe that open source culture is natural to computer science education, promoting a curriculum based on collaborative development. Integrating open source concepts and practices improves undergraduate computer science teachings and eventually helps computer science students align better with professional world.

## II. Impacts of Open Source on CS Education

Based on our literature survey, we have extracted the following list of items that researchers have claimed will be enhanced by using open source in education in general and CS-education in particular [3-8].

1- Collaborative Team Learning: For students studying computer science, working with peers in a team is very critical. Open source naturally promotes collaboration, and adopting the collaborative model of open source to software development courses seems to be very useful to students. For example, fairly complex software requires many iterations and levels of testing. In such cases, open source communities can help in providing testing environment as well as interested testers to test the new software [7]. In addition, students can benefit from open information sharing [5]. In [4], the 'Open Source Collaboratory' model is presented so that students are able to configure their computing systems themselves as required by their courses without any system administration rules and contributions [4].

2- Version and Source Control: Learning, studying, and understanding software development process require students to have access to earlier versions of computer programs to investigate and track changes in them [3]. Software development includes many different repetitions and iterations, which implies several different versions of the code (which in turn could include many different revisions). Version control systems (VCS) are used for this purposes and it is a necessary skills for all software engineers to be able to work with such tools. In modern software engineering, distributed version control systems (DVCS) such as Git servers are used which provide additional features compared to traditional VCS's (aka centralized VCS) such as working offline, managing new revisions and versions much easier, and manage changes in the "line" level as opposed to committing the whole file every time even for very minor changes [14].

3- Integrated Development Environment (IDE): Independency from any commercial vendor for the development environment since it comes with extended documents [3,6].

4- Free Software: Helping Students understand the differences in initial cost as well as ongoing and support cost when using open source software [3,4,7,8].

5- Modifiability: Open source tools are easier to modify and customize free of charge [3-8].

6- Complete Software Development Process: Students are prepared for fully engaged software engineering tasks and projects [3].

7- Real-life Projects: Students can learn better how real software solutions look like based on customer needs [7].

8- Software Testing: A variety of testing environments are introduced to students [8].

9- Environment Setup and Configuration: Once students learn the environment, it is going to be a fast process for them to use it [5-7].

## III. Proposed Open Source Labs for Software Engineering

In this section, we explain the list of lab activities that we developed. It should be emphasized again that we believe the main goal of software engineering course is to teach students the practical software development process, which requires several skills sets. These categories can be listed as knowledge and skills about the development environment and corresponding operating system, tools such as resource management, as well as communications and also the software engineering process skills. Based on these categories, we designed eight labs. Figure 1 summarizes the software engineering skills categories, which are mapped into our designed labs as listed in the following.

- **Lab 1 – Set up the Development Environment**

  In this lab students build their virtual machines running a Linux Operating System such as Fedora. The objective of this lab is to help them understand the attributes of a collaborative development envrionment.

- **Lab 2 – Get connected - How does the Open Source Community communicate?**

  In this lab students become familiar with different communication tools (such as iRC, TitanPad, Skype, mailing lists etc.) that are freely available for everyone and are commonly used by open source community. In addition, they can explore how the team can interact with each other regarding the project issues. Students are also able to find out how different open source software development communities (in particular HFOSS) use these tools to perform efficient communications among their members.

- **Lab 3 – Exploring Linux: Overview of Linux Environment and Commands**

  This lab helps students get familiar with Linux and how to navigate around inside Linux operating system. Since Linux is the natural platform for open source development, students has to acquire basic skills and knowledge of this

operating system and in particualr working with command lines. The lab activities mainly includes several different scenarios and corresponding command line interface (CLI) commands. Simple shell scripting for customizing environment is also covered.

- **Lab 4 – GIT 101**

  In this lab, students learn how to set up Git server for cloning and version control. At first, students learn about the features of Git servers and they set up their own repository and version control in the previously set environment. Here, the idea is to show students how each developer works on her own local development environment that is in sync with the shared code on the Git server and therefore, all the team members are in sync with each other. Consequently, each team member installs and configures a local Git repository for projects [13].

- **Lab 5 – Building a Cloned Module**

  In this lab we describe how to clone an open source module such as Open MRS, Gnome Music, etc. Students learn how to build the cloned modules by replicating the dependencies properly through using the Git server. This is the simulation of creating a local development envrionment for each team member.

- **Lab 6 – Modeling an existing Open Source Software using Structured and Object Oriented Analysis**

  In this lab, students use structured modeling such as data flow diagram, functional decomposition, etc. as well as UML diagrams to reverse engineer the cloned module. The idea in here is to help them work with modeling open source tools [15] to model existing open source tools. In general, open source software do not have a very solid modeling documentation. In [11], authors studied different research questions on how and why Open Source developers use diagrams. Several different options are discussed such as using diagrams for reverse engineering of the projects with no or little documentation.

- **Lab 7 – Requirement Analysis**

  Goal of this lab is to help studnets learn how to work with a fairly large codebase. The idea is to simulate a newly hired team of developers to work with existing software and make some enahncements to it. Therefore, a scenario will be given to them to make some customiztion to the cloned software from last lab. For example, students have worked with OpenMRS, Agilefont, GnomeMusic, and etc. and have made some modifications to them based on the extracted requirements.

  The requirements are summarized by students as a list of product backlogs coming from user stories. They are written very clearly in detail explanations to avoid misiterpretation in the latest phases of system development. In this particualr lab they learn how to analyze user stories, validate them, and eventually document the requirements.

- **Lab 8 – Developing/Implementing/Testing the Proposed New Features**

  In this lab, students will design, develop, and implement the proposed new features to be added to the current system. This lab simulates the concept of enhancement rather than bug fixing.

As mentioned before, figure 1 maps our proposed labs into the necessary sets of skills and knowledge for learning software engineering.

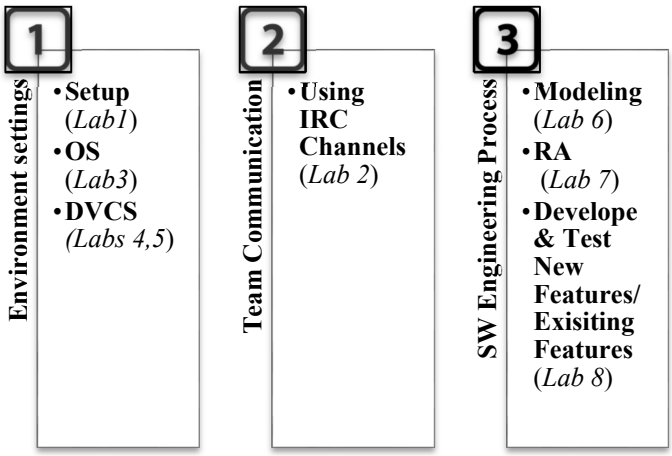


Figure. 1. Mapping of the proposed lab activities into software engineering knowledge and skills

## IV. EVALUATION

As explained earlier, the main purpose of the proposed activities is to teach students the practical aspects of software engineering including environment set up, source and version control, bug report and bug fixing. In this way, students perform the ordinary tasks of standard software engineering. They will be asked certain questions before working with the proposed labs. This is the pre-test data collection. Students will also be asked some questions after they finished the lab activities. This data is referred to as the post-test data. Next, different hypothesis tests are applied on the collected data as explained in the extensive study in [9, 10]. The main goal is to measure the impact of such activities on students' learning.

The focus of the works in [9] and [10] is on Humanitarian Free and Open Source Software (HFOSS) and its impact on student computational thinking and learning, in addition to keeping them motivated to graduate from the major. They studied the students' participation in HFOSS from different schools (with different sizes and focus). The impacts of HFOSS are reported in [9] and [10] using quantitative analysis of Likert survey items. The results showed mixture of positive and negative results. The positive results proved that students who worked on HFOSS projects improved their computing skills and felt that they could comfortably work with professionals on real world projects.

Negative results included an overwhelmingly decrease in students' confidence dealing with the complex and large real-world project [9, 10].

In our study, we followed the same type questionnaire as in [9] and [10] and also introduced students to HFOSS projects. Our participants were mainly new graduate students and some upper division undergraduate students.

### A. *Dependent Group Test of Hypothesis: Pre-Post Test*

The goal of the tests here is to determine whether the there has been any improvement on the opinion of students related to the following statements before and after they did the proposed lab activities.

Here is the list of survey questions.

1. I can describe the impact of project complexity on the approaches used to develop software.
2. I can describe the impact of project size on the approaches used to develop software.
3. I can use a software process to develop an HFOSS project.
4. I am confident that I can maintain an HFOSS project.
5. I can describe the drawbacks and benefits of FOSS to society.
6. I can use all tools and techniques employed in my project.
7. I can participate in an HFOSS development team's interactions.
8. Participation in an HFOSS project will improve my understanding of how to behave like a computing professional.
9. I feel confident about working with computing professionals.

For this purpose, the hypothesis test shown below is applied to the above list of survey questions.

- $H_0$: Median of the students' responses after the lab <= Median of the students' responses before the lab
- $H_1$: Median of the students' responses after the lab > Median of the students' responses before the lab

We used the *Wilcoxon Mann-Whitney test* which is a non-parametric statistical hypothesis test used when comparing two related samples. The result of every single of the above tests was to reject the null hypothesis which confirmed the effectiveness of the method. Therefore, our results comply with the positive findings in [9] and [10] with higher rate of positive responses from students.

### B. *1-Sample Test of Hypothesis*

In addition, we broaden our study beyond the impact of HFOSS and looked into the concept of using large and established projects in understanding of the whole software engineering process. For our evaluation, we are using mainly the post-test questions related to software engineering learning. In our analysis, we are using 1-sample Wilcoxon Test comparing the opinion of the students to the average value of 3 since students' responses' include 1 to 5 ranging from strongly disagree (1), disagree (2), neutral (3), agree (4), and strongly agree (5). The other two answer options are 'not applicable (6)' and 'don't know (7)'. We have filtered these two answers.

The goal of the tests in this part is to determine whether the median of the students' responses to a particular question differs from the average of the possible responses (1 to 5), which is 3. This could be an indicator that the open source lab activities have been effective.

For this purpose, the hypothesis test shown below is applied to the following list of survey questions.

- $H_0$: Median of the students' responses <= 3
- $H_1$: Median of the students' responses > 3

The result of every single of the above tests was to reject the null hypothesis, which confirmed the effectiveness of the method.

## V. FUTURE WORK

As mentioned, students can use open source resources as a major learning tool. Even though some researchers reported that working with OSS could be challenging for beginners [8], we think students need to start learning about it early in their education. Ultimately, such early exposure needs to be studied as far as its impact throughout their computer science education.

## VI. CONCLUSION

Using open source projects in software engineering course is showing many positive impacts in learning practical software engineering conepts. Even though some educators have warned about features of open source which seem very useful to education, but could be difficult for both instructors and students for teaching and learning [2], students can learn many things from OSS. They can start learning about the communication methods of open source community followed by getting familiar with the available free tools which is the basis of most similar commercial products. In addition, they have a chance to work with already developed large software to enhance their features including fixing bugs as well as adding new features to it. Another valueable experience student could earn by using open source is using the open source reverse engienering and modeling tools. Since they work with very large codebase, modeling is much more challenging and therefore, it is a great chance for them to see different models of large codebases.

## VII. References


[1] J. Dionisio, et. al., An open source software culture in the undergraduate computer science curriculum, Newsletter, ACM SIGCSE Bulletin, Volume 39 Issue 2, June 2007, Pages 70-74.

[2] H. Ellis, R. Morelli, and G. Hislop, Support for Educating Software Engineers Through Humanitarian Open Source Projects, 21st Annual Conference on Software Engineering Education and Training, Charleston, SC.

[3] T. Leinonen, J. Pietarila, and G. Kligyte, Developing Open Source Software for Educational Sector – Case: Future Learning Environment, EurAsia-ICT 2002, Shiraz-Iran, Oct. 29-31, Pages 131-137.

[4] Jeff Wright, et. al., Collaboratory: An Open Source Teaching and Learning Facility for Computer Science and Engineering Education, In Proceedings of the 2007 International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS), Las Vegas, Nevada, June 25-28.

[5] Joshua M. Pearce, The case for open source appropriate technology, Environ Dev Sustain (2012) 14:425–431.

[6] Erik J. Martin, Open Source: Is that Right for you? Jan/Feb. 2013 ECONTENT, Pages 14-19.

[7] Matthew J. Hawthorne, Software engineering education in the era of outsourcing, distributed development, and open source software: Challenges and opportunities, Proceedings of the 27th Int'l Conf. on Software engineering, 2005.

[8] S. E. Lakhan and K. Jhunjhunwala, Open Source Software in Education, EDUCAUSE Quarterly, Nov. 2008.

[9] G. Hislop, H. Ellis, S. Pulimood, B. Morgan, S. Mello-Stark, B. Coleman, C. Macdonell, "A Multi-Institutional Study of Learning via Student Involvement in Humanitarian Free and Open Source Software Projects" ACM ICER 2015 Aug 9-13 Omaha, Nebraska.

[10] H. Ellis, G. Hislop, S. M. Pulimood, B. Morgan, and B. Coleman, 2015. Software Engineering Learning in HFOSS: A Multi-Institutional Study, In Proceedings of the 122nd Annual ASEE Conference and Exhibition, Seattle, WA.

[11] K. Yatani, E. Chung, C. Jensen, K. N. Truong, Understanding how and why open source contributors use diagrams in the development of Ubuntu. CHI '09 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. Pages 995-1004.

[12] L. Haaranen, T. Lehtinen, Teaching Git on the Side – Version Control System as a Course Platform, Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education, Pages 87-92.

[13] D. Burdge , S. Jackson, GIT 101: Foundations for a Common Workflow to Contribute to HFOSS, Journal of Computing Sciences in Colleges, Volume 31 Issue 6, June 2016, Pages 18-20.

[14] C. Brindescu, M. Codoban, et. al. ,How Do Centralized and Distributed Version Control Systems Impact Software Changes?, ICSE '14, May 31 - June 7, 2014.

[15] T. Buchmann, owards Tool Support For Agile Modeling:Sketching Equals Modeling, 2012, ACM 978-1-4503-1804.