# Supporting the Learning Process of Open Source Novices: An Evaluation of Visualization Tools

Yunrim Park, Carlos Jensen
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR 97331
+1-541-737-2555

{parkyu, cjensen}@eecs.oregonstate.edu

## ABSTRACT

Exposure to Open Source Software (OSS) projects provides students with opportunities to experience “real-world” software development processes and practices, including the opportunity to work with large, existing, and complex code-bases and development communities. Working with OSS also poses challenges. In this paper, we look at some of the challenges students may face when learning about and joining an OSS project. We investigate how visualization tools can help streamline the initial learning process, and present results from a controlled experiment.

## Categories and Subject Descriptors

K.3.2 [Computer and Information Science Education]: *Computer science education*

## General Terms

Performance, Experimentation, Human Factors.

## Keywords

Open Source Software (OSS), Computer Science education, Code Visualization, Empirical study

## 1. INTRODUCTION

Integrating Open Source Software (OSS) projects and development practices into a Computer Science (CS) curriculum has long been a goal for many. With the growing popularity, diversification, and appeal of OSS among both industry and end-users, the use of OSS in a CS curriculum is increasingly important to ensuring students gain real-world, hands-on experience and skills that are likely to be appreciated by future employers [3, 8, 10, 12, 21]. From an educators’ perspective, OSS has a certain allure in its potential to make the curriculum more interesting, challenging, and personally meaningful, which might attract more students to CS [6, 13].

There is no question that students can benefit from engaging in OSS development, and earlier experiments have reported positive learning outcomes [1, 5, 6, 14]. Educators should however be aware that integrating OSS development into a curriculum presents a set of problems. Most of the discussion so far has centered around developing and deploying new projects and assignments, and reporting on the resulting improvements. Little attention has been directed to reflecting on challenges or failures. Only a few educators have reported on the problems they experienced [3, 12, 16], and these are mostly presented as warnings to others rather than reflections on how to overcome or avoid pitfalls.

This paper attempts to shed some light on the most common challenges encountered when integrating OSS into a CS curriculum from a student’s perspective. Our goal in doing so is to help educators think about both the potential benefits and the pitfalls associated with such a curriculum change. Furthermore, we discuss how to empower students to deal with the challenges associated with joining and exploring an OSS project through the use of existing code visualization aimed at lowering the barriers for entry into the world of OSS.

In the following section, we discuss reasons for integrating open source into a CS curriculum, as well as the challenges students may face as a result. Section 3 describes our experiment, which measures the benefits of using visualization tools in the learning process. The results are presented in Section 4. We conclude with feedback from the students who participated in the study and our suggestions for educators.

## 2. BACKGROUND

Perhaps the biggest driver for the recent move to integrating Open Source in CS education is the desire to address some of the limitations of conventional CS curriculums in providing students with real-world experience. OSS projects and tools are generally available at little or no cost, and they can share many of the same characteristics commonly found in real-world software development projects; large code-bases, large distributed development teams, long development histories, etc. Exposure to OSS development will provide students with experience in solving more “realistic” problems in a more “realistic” setting. Such experience is likely to be more meaningful and immediately applicable, something that will place students in a better position when they look for jobs. Beyond the purely technical, through their participation in project activities and interaction with community members, students will learn organizational, interpersonal, and mentorship skills that are important in professional advancement [19].

Unfortunately, the OSS model also presents a set of challenges to educators - often directly related to the opportunities presented. As an example, students who lack experience or confidence in programming may have trouble dealing with large code bases.

[15]. Likewise, though adding to the realism, understanding the inner workings of the poorly documented code often found "in the wild" can be daunting [14]. Pedroni et al.'s controlled study on students' motivation working on a conventional class project vs. an open source project showed that the open source group showed a decrease in their level of interest, while the other control showed an increase, indicating that students often struggle with "the first hurdle of basic understanding and involvement"[16]. Assignments or coursework that involve students engaging or participating in actual OSS development activities such as fixing a bug or extending an existing feature may 1) be difficult to identify before a course starts (as these may be addressed before students have an opportunity to do so), and 2) be difficult for an instructor to accurately a priori judge the difficulty or time investment required by students. As Toth observes, "The biggest challenge in practicum development is to formulate real-world problems that engage the students and can be solved within five to six months." [21]

Working on an OSS project often requires one to learn about its history, conventions, design decisions, current status, and goals, knowledge of which may be crucial to performing activities in OSS communities, or having patches accepted. As a first step to joining a project, newcomers are often encouraged to learn about the project from publicly available resources such as mailing lists, IRCs, bug trackers, and source code repositories. It is the norm for most OSS contributors and users to communicate ideas and share using these resources. Therefore, these resources usually contain comprehensive information about the project, often taking the place of a requirement document, manual, or other more definitive and structured reference source. Learning from such artifacts, however, can be time-consuming because of the amount of information accumulated over the life of the project and the sometimes ad-hoc nature of its organization.

While building social and collaborative skills through interaction with community members is one of the potential benefits of using OSS projects in a CS curriculum, it is the least explored or documented by educators. This is in part because the development of these skills is difficult to measure, part because it is difficult to accomplish in the course of one term, and part because this requires support from the community in question. Although OSS communities are open and welcome newcomers, they may not be very enthusiastic about guiding large numbers of students through the learning process, unless these students will be long-term contributors. In addition, students may not feel comfortable asking questions or participating in public discussions for fear "flaming" or "losing face" due to their lack of experience. Krogstie after his experiment teaching software engineering practices through collaboration with an OSS community says "Student SE teams may hesitate to embark on interaction over technically challenging issues in a type of community unknown to them, the students being in doubt about their own skills and credibility" [12]. A student's reflection on his learning experience in OSS added: "It's hard for us to say that our design could possibly be better than the original programmers" [3].

To address these challenges, researchers and educators have tried a number of ideas and approaches. Some include limiting the scope of activity (e.g. no community interaction, no actual contribution to the community) [10, 18], using "education-friendly" projects [14], and coordinating with the project community prior to deploying new curriculum [6]. These limited approaches are often inevitable given the time constraint of academic terms and necessary to reduce the steep learning curve and the pressure on students. On the other hand, limiting participation can lead to limiting the benefits of integrating OSS projects into curriculum. A better approach may be organizing participation around a sequence of courses, increasing the level of difficulty and participation as students become more familiar and experienced with OSS development practices [5, 16, 21].

One fundamental challenge, however, still remains; learning about technical and social aspects of the project is left to individual students. This paper reports on our efforts to investigate how to empower students to overcome these barriers themselves. We use code visualization tools to support the learning process and present empirical evidence on the value of these to students first approaching an OSS project. We also present a set of exploratory questions that can help students find their bearings more quickly.

## 3. EXPERIMENT

### 3.1 Design

The goal of our study was to determine the potential benefits of existing code visualization tools for students new to OSS development. To this end, we conducted a three-group between-subjects experiment where participants were asked to find specific information about a given OSS project. Table 1 shows the resources and tools provided to each group.

**Table 1. Control vs. Experimental Groups**

| Group | Default Resources | Visualization Tools | |
|---|---|---|---|
| | | Change History | Code Structure |
| A | • Website & tools in SourceForge.net<br>• GanttProject's own website<br>• Eclipse IDE | N/A | |
| B | | Version Tree* | Creole* |
| C | | Augur [4] | SA4J* |

* Version Tree: http://versiontree.sourceforge.net/
* Creole: http://www.thechiselgroup.org/creole
* SA4J: http://www.alphaworks.ibm.com/tech/sa4j

For the experiment, we selected an actual OSS project called "GanttProject" (http://ganttproject.biz/) from SourceForge.net based on a set of criteria including programming language (Java), code repository (CVS), number of contributors (10+), size of code base (10K+ lines of code), age (2 years+), activity and maturity (90%+ and Production/Stable on SourceForge.net). These criteria ensured that the project was compatible with the visualization tools available and had enough activity and artifacts to make the tasks challenging but not overwhelming.

SourceForge.net provides a set of standard tools and infrastructure common to most OSS projects, which served as a "default" for the experiment and were available to all groups. The experimental groups, B and C, had two additional visualization tools (Table 1). Each pair of the tools serves the same purposes but differs slightly in their feature sets (Table 2 and 3). The biggest difference between Group B and C was that the tools for Group B were integrated into the IDE (i.e. Eclipse) while those for Group C were standalone applications. The two groups were set up to see if the integration of the tools in an IDE affects performance.

**Table 2. Version Tree vs. Augur**

| | Version Tree | Augur |
|---|---|---|
| **Application type** | Eclipse plug-in | Standalone |
| **Supported system** | CVS | CVS, Subversion |
| **Features** | Displays revision history in tree view, supports code browsing and easy switching between revisions | Displays revision history in various ways (line-oriented view, line charts, bar charts), provides information about author-activity, time-activity, and other statistics |
| **Visualization units** | Individual files | A wide range from line-level to an entire project |

**Table 3. Creole vs. Structural Analysis for Java (SA4J)**

| | Creole | SA4J |
|---|---|---|
| **Application type** | Eclipse plug-in | Standalone |
| **Supported lang.** | Java | Java |
| **Features** | Displays code structure and relationship between elements in different views (nested, class interface hierarchy, call graphs, fan in/out, etc.) | Displays code structure and relationships between elements both in graphic and in text, provides analytic information (antipattern detection, stability analysis, package analysis, etc.) |
| **Dependency information** | Between packages and within package dependencies, nested view, call graphs | Local/Global dependencies, What if view (propagation of change) |

## 3.2 Tasks

The selection of tasks was a two-step process; first, identifying the information needs of those who wish to work on an OSS project as a code contributor, and second, defining concrete tasks that would force participants to seek this information from a project.

We reviewed previous studies on the information needs in collaborative software development and selected the four [7, 9, 11, 17] most relevant to our work. These studies examined collaborative software development and/or maintenance processes and identified information critical to performing these tasks and collaborating with other team members. We listed the different types of information identified in our four sources and grouped these into six categories: (1) Main contributors and their expertise, (2) History of changes, (3) Structure of code and relationship between different elements, (4) Impact of a change, (5) Control flow and execution behavior, and (6) Source code details. Using these categories, we defined tasks which would mimic real-life learning situations without overwhelming participants. From these we derived four tasks for participants to do. Participants had 15 minutes to complete each task. The questions in each task are given here to serve as a template for similar exercises for students exploration:

*Task1. Find information about active contributors and change history at the project-level*

1. Who are the most active code contributors? Name three.
2. Where in the code are they working?
3. How large is the code base of the project?
4. Find information about the changes to the CVS repository over the last 12 months.

*Task2. Find collaborators and change history at a package-level*

1. Find the contributors who know about the package. Name three of them.
2. When was the most recent change made in the package and who made the change?
3. How has the package been changed since it was first created in terms of size?
4. What should you need to know to make a change in the package and to submit it?

*Task3. Understand code structure and relationship between elements within a package*

1. How many classes does <package> have? And how many interfaces?
2. Find the classes that are heavily interacting with other classes in the package <package>? Name two of them.
3. Suppose you want to change the class <class>. Which classes of the package <package> will be directly affected by this change?
4. Which of them is the most dependent on <class>? Describe the dependency.

*Task4. Understand relationship and dependencies between two packages*

1. How are the two packages related?
2. Which class of the package <package> is the most dependent on the package <package2>?
3. Which class of the package <package2> is the most dependent on the package <package>?
4. How and where would you find the information that might be useful in understanding the existing code?

Version Tree and Augur visualize the history of changes made to code base, including core contributors, their expertise, and the amount of contribution (Table 2). These tools were potentially helpful in completing Task 1 and Task 2. Creole and SA4J visualize code structure and relationships between different elements (Table 3). These supported doing Task 3 and Task 4.

## 3.3 Participants

27 participants (18 males and 9 females) were recruited from the School of Electrical Engineering and Computer Science at Oregon State University. All had at least 2 years of experience with Java programming. Participants were randomly assigned into the three groups, which were gender-balanced. Prior to the experiment, they received a quick introduced to the environment, provided a brief description of the OSS project, as well as a refresher on the Eclipse IDE. Those assigned to an experimental condition were given a short tutorial on the extra tools, explaining the purpose of the tools and demonstrating their key features.

## 4. RESULTS

Two researchers graded replies on a scale from 0 to 10 per question. The mean scores are shown in Figure 2. The inter-rater reliability of the graders showed a very high positive correlation (Pearson's correlation coefficient r=0.8464), supporting the validity of scores. We also looked at how many people in each group managed to complete each task (i.e. answering all 4 questions). Task completeness (time) and correctness (scores) between groups were analyzed using the Kruskal-Wallis non-parametric test because there was no assurance of normality in the distribution. We also looked at the data from pre- and post-session questionnaires and comments made during the session.

Participants in Group B and Group C predominantly used the treatment tools and relied little on the default resources. The only exception was Group B in the first two tasks. Participants used the default resources more than Version Tree, and did not show a preference to the tool because of its inability to visualize aggregated information about changes in the package or higher level. Thus the rest of the result section omits the comparison between Group A and Group B in the first two questions.

## 4.1 Task Completeness

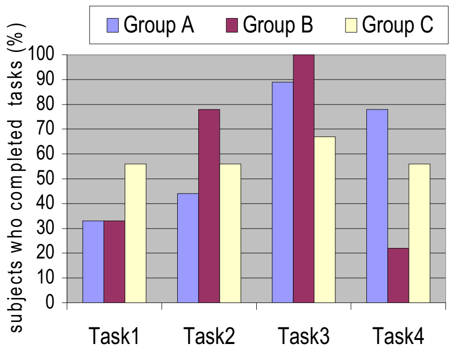

**Figure 1. Number of participants completing each task**

Figure 1 shows the number of participants who completed the entire task in the time allotted. Group C performed better in the first two tasks than Group A, but did worse in the last two tasks. Group B performed better than Group A in Task 3, but did significantly worse in Task 4 because of the tools' ("Creole") representation of different types of dependencies. While dependencies between elements within the same package (Task 3) were easy to perceive from the view generated by the tool, dependencies between two different packages (Task 4) were not as intuitive and required interaction with the view. This hurdle contributed to the different completeness between two tasks.

## 4.2 Correctness of Answers

Figure 2 shows the mean task scores of each group (correctness). Group C scored higher than Group A throughout the tasks. Group B performed better than Group A in Task 3 and Task 4. Statistical analysis of task correctness (i.e. task scores) between groups (*Kruskal-Wallis rank sum: two-sided, df=2*) showed significant differences in the first three tasks; (1) Task1: $\chi^2=8.7496, p<0.013$, (2) Task2: $\chi^2=6.3274, p<0.043$, (3) Task3: $\chi^2=7.9851, p<0.019$.

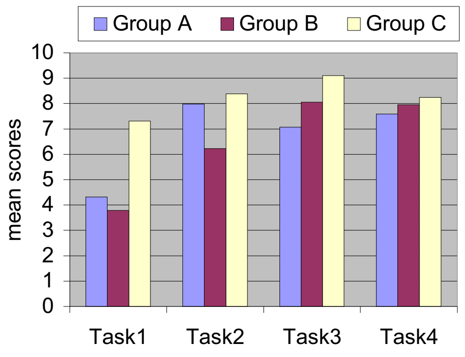

**Figure 2. Mean scores of each task**

High mean score and task completion of Group C in Task 1 and Task 2 suggests that the visualization tool ("Augur") improved both the efficiency of finding information and the quality of the information. The dominant tool for Task 3 and Task 4 ("SA4J"), on the other hand, improved the quality but did not help much on the efficiency mainly due to the complexity of the tool itself. The visualization tool ("Creole") helped Group B with Task 3 in both completeness and correctness. Contrary to our expectations, we did not see clear differences in performance between plug-ins and stand-alone applications.

Overall, these tools made it easier to find information about a project by presenting large amounts of data in an accessible form. Although some of the tools confused participants, resulting in lower completion in Task 3 and Task 4, this was not surprising given the short training and experience with the tools participants had prior to the experiment. Participants' comments also corroborated that they did not feel the tools were very difficult to use and preferred using them over the default resources.

## 5. DISCUSSION

Our controlled experiment showed the potential benefits of introducing visualization tools to students learning about a new OSS project, even when time is limited. Using tools like Augur and Creole helped participants find information efficiently which otherwise would require intensive work. As a result, more participants in corresponding groups completed their tasks. The quality of information obtained was generally higher in experimental groups than that in the control group. Difference in performance between the control group and the experimental groups came mostly from those questions requiring extensive search and analysis (e.g. *Where in the code are the active code contributors working?, How has the package been changed since it was first created in terms of size?)*.

Although complete information about past activities and the current status of the project was available from the default resources (e.g. forums, mailing lists archives, bug repositories), participants seemed at a loss for where to start looking for information and overwhelmed by the amount of information they would need to go through to find their answer. Besides, text-based tools did not provide an efficient way of getting "a big picture" view of the project. One participant' commented about the advantage of the code visualization tools exemplifies this: *"I could look at the source code. But then... if I had a bigger problem, my solution... finding a solution would directly be dependent on my time and not using the tools. Because looking at the source code, which is manual work…"*

Participants' comments also point to challenges in learning and low confidence in their own abilities. One of the participants said, *"The SourceForge repository can be made more interactive and user-friendly. This would make it look less geeky/daunting for first-timers."* Some participants, despite being competent developers, sought to place blame on their own inexperience when they failed: *"I'm not so familiar with open source…", "for me, I haven't worked on open source... so I don't know where they have this statistic."* Such low self-efficacy may affect students' motivation and have a negative effect on their learning [2].

The tools and resources commonly found in OSS projects and most teaching environments may not be sufficient for students to learn about an OSS project in a limited amount of time. We believe providing students with friendlier environments will lower their learning curve and make their first experience working on an

OSS project more enjoyable. For instance, the tools we used in the experiment received positive comments from participants. They learned how to use the tools fairly quickly and preferred using them over the default resources.

Although a number of visualization tools have been developed [20], to date we have seen little mainstream adoption of such tools and techniques in CS curricula. This may be because conventional class projects and assignments are small and simple enough to be done without tools, or educators were reluctant to introduce tools worrying that students might become dependent on them. While the best way of getting students engaged with OSS projects may be letting them explore and spend time on the projects as most OSS contributors would, it may not be feasible because of time constraint, taking time away from other curricular objectives. We therefore believe that curricular change should be accompanied by efforts to create a more supportive learning environment.

## 6. CONCLUSION

Our work shows that visualization tools can be beneficial to those learning about an OSS project and help students find information about the given OSS project more efficiently and effectively. Providing more efficient ways to handle a large amount of information and understand code can lower the learning curve and information overload that students may experience when they start working on an existing OSS project. Students' positive feedback on the tools and low confidence attributed to their inexperience with OSS development suggest that the incorporation of visualization tools will help students and enhance their learning experience.

## 7. ACKNOWLEDGMENTS

We thank the students who participated in this study. We also thank Eunyoung Chung for her contribution to this work, and the HCI research group for their feedback and help.

## 8. REFERENCES

[1] Allen, E., Cartwright, R., and Reis, C. 2003. Production programming in the classroom. SIGCSE Bull. 35, 1 (Jan. 2003), 89-93.

[2] Bandura, A. (1994). Self-efficacy. In V. S. Ramachaudran (Ed.), Encyclopedia of human behavior (Vol. 4), 71-81

[3] Carrington, D., Kim, S.-K., 2003. Teaching software design with open source software. Frontiers in Education, 2003. 33rd Annual, vol.3, no., pp. S1C-9-14 vol.3, (Nov. 2003), 5-8.

[4] de Souza, C., Froehlich, J., and Dourish, P. Seeking the source: software source code as a social and technical artifact, Proc. of the 2005 Int. Conf. on Supporting group work, Sanibel Island, FL, USA, 2005, 197 – 206.

[5] Dionisio, J. D., Dickson, C. L., August, S. E., Dorin, P. M., and Toal, R. 2007. An open source software culture in the undergraduate computer science curriculum. SIGCSE Bull. 39, 2 (Jun. 2007), 70-74.

[6] Ellis, H. J., Morelli, R. A., de Lanerolle, T. R., Damon, J., and Raye, J. 2007. Can humanitarian open-source software development draw new students to CS?. SIGCSE Bull. 39, 1 (Mar. 2007), 551-555.

[7] Erdos, K. and Sneed, H.M. 1998. Partial Comprehension of Complex Programs (Enough to Perform Maintenance), Proc. of the 6th Int. Workshop on Program Comprehension, Ischia, Italy, 1998.

[8] Fuhrman, C. P. 2006. Appreciation of software design concerns via open-source tools and projects. In 10th Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts, at 20th European Conf. on Object Oriented Programming (ECOOP), Nantes, France, July 2006.

[9] Gutwin, C., Penner, R., and Schneider, K. 2004. Group Awareness in Distributed Software Development. CSCW'04, November 6-10, 2004, Chicago, IL, USA.

[10] Hepting, D. H., Peng, L., Maciag, T. J., Gerhard, D., and Maguire, B. 2008. Creating synergy between usability courses and open source software projects. SIGCSE Bull. 40, 2 (Jun. 2008), 120-123.

[11] Ko, A., DeLine, R., and Venolia, G. 2007. Information Needs in Collocated Software Development Teams, Proc. of the 29th Int. Conf. on Software, 2007.

[12] Krogstie, B. R. 2008. Power through brokering: open source community participation in software engineering student projects. In Proc. of the 30th Int. Conf. on Software Engineering. ICSE '08. ACM, NY, 791-800.

[13] Layman, L., Williams, L., and Slaten, K. 2007. Note to self: make assignments meaningful. In Proc. of the 38th SIGCSE Technical Symposium on Computer Science Education. SIGCSE '07. ACM, NY, 459-463.

[14] Meneely, A., Williams, L., and Gehringer, E. F. 2008. ROSE: a repository of education-friendly open-source projects. In Proc. of the 13th Annual Conf. on innovation and Technology in CS Education. ITiCSE '08. ACM, NY, 7-11.

[15] Patterson, D. A. 2006. Computer science education in the 21st century. Communications of the ACM 49, 3 (Mar. 2006), 27-30.

[16] Pedroni, M., Bay, T., Oriol, M., and Pedroni, A. 2007. Open source projects in programming courses. SIGCSE Bull. 39, 1 (Mar. 2007), 454-458.

[17] Sillito, J., Murphy, G.C., and De Volder, K. 2006. Questions Programmers Ask During Software Evolution Tasks, Proc. of the 14th ACM SIGSOFT Int. Symposium on Foundations of Software Engineering, OR, USA, 2006, 23-34.

[18] Sowe, S.K., Stamelos, I. 2007. Involving Software Engineering Students in Open Source Software Projects: Experiences from a Pilot Study. Journal of Information Systems Education, Vol. 18(4), 425-436.

[19] Spinellis, D. 2006. Open Source and Professional Advancement. IEEE Software 23, 5 (Sep. 2006), 70-71.

[20] Storey, M. D., Čubranić, D., and German, D. M. 2005. On the use of visualization to support awareness of human activities in software development: a survey and a framework. In Proc. of the 2005 ACM Symposium on Software Visualization. SoftVis '05. ACM, NY, 193-202.

[21] Toth, K. 2006. "Experiences with Open Source Software Engineering Tools," IEEE Software, vol. 23, no. 6, (Nov/Dec, 2006), 44-52