Lessons Learned from Teaching Open Source Software Development

Becka Morgan, Carlos Jensen

Lessons Learned From Teaching Open Source Software Development

Becka Morgan[1], Carlos Jensen[2]
[1]Western Oregon University, Monmouth, OR, USA
morganb@wou.edu
[2]Oregon State University, Corvallis, OR, USA
cjensen@eecs.oregonstate.edu

Abstract. Free/Open Source Software allows students to learn valuable real world skills and experiences, as well as a create a portfolio to show future employers. However, the learning curve to joining FOSS can be daunting, often leading newcomers to walk away frustrated. Universities therefore need to find ways to provide a structured introduction to students, helping them overcome the barriers to entry. This paper describes two courses taught at two universities, built around a Communities of Practice model, and the lessons learned from these. Suggestions and insights are shared for how to structure and evaluate such courses for maximum effect.

Keywords: Free/Open Source Software, Education, FOSS

1 Introduction

Free/Open Source Software (FOSS) is an increasingly important part of the computing eco-system [1]. Teaching students how to participate in FOSS projects not only provides them with meaningful and highly marketable hands-on experience, it also potentially helps ensure FOSS communities have enough qualified developers to draw from to meet their needs. Supporting a growing and vital FOSS eco-system requires us to grow the pool of potential contributors. While working things out from first principles by yourself has been the traditional way of learning how to contribute to FOSS, this is a very inefficient model, and unlikely to meet growing needs.

Newcomers to FOSS often have a difficult time finding an entry point into a project. Barriers to entry include documentation that is incomplete and/or not up to date, no response from community members when questions are asked, and the need to learn a new set of tools in order to participate[2]. While braving the learning process and overcoming the many obstacles on your own is seen as a badge of courage, we believe there is ample room for improvement, and a need for a guided and more structured learning experience.

adfa, p. 1, 2011.

Beyond the economic/market arguments, as educators we have a duty to look towards the needs of our students, and how to best prepare them for the jobs of tomorrow. Experience with, and the ability to participate in FOSS development is becoming a critical skill, as companies increasingly adopt FOSS[3], and even when not using FOSS, look for FOSS experience in their hiring. According to David Heinemeyer Hansson, a partner at the software development firm 37signals, "Open source is a golden gift to the hiring process of technical people. It reduces the risk enormously by allowing you to sample candidates over a much longer period of time" [4].

As FOSS projects give students a unique way of developing real-world skills and experience, as well as providing prospective employers with tangible proof of their skills, it is important for universities to develop class-room models to teach more students how to participate in FOSS. Some universities have been experimenting with such courses, and this paper describes two different approaches to teaching students how to navigate the joining process and contribute to a FOSS project. One instructor focused on building on a collaborative "communities of practice" model, while the other on a more traditional classroom model. This paper compares and contrasts the two models different models, each used twice at different universities. It discusses the outcomes, lessons learned, and provides guidance to those contemplating developing their own courses on this topic.

2 Related Work

Work has been done looking at how to incorporate FOSS into CS curriculum, and even how to use FOSS projects as a way of recruiting students looking to have more real-world impact. One such effort is the Humanitarian FOSS project (HFOSS) [15]. This curriculum focuses on creating enthusiasm and engagement around participation in HFOSS, primarily among non-traditional CS students. Through this effort they seek to provide compelling use-cases and activities that will motivate a wide range of student groups, and be able to effectively leverage diverse backgrounds and experiences to solve real-world problems [9, 16].

The HFOSS project has also influenced other curriculum efforts, most notably the "Professors Open Source Summer Experience" (POSSE), sponsored by Redhat [5]. This effort is aimed at teaching college professors about FOSS, about FOSS tools, and how to introduce FOSS into their curriculum. This paper focuses on lessons learned from using the "go it alone" model, then incorporates information gained from attending POSSE to outline an improved curricular model designed to support students in overcoming the barriers to entry. In addition, there have been a number of efforts designed to produce curriculum around FOSS, such as the Teaching Open Source web community [6], which shares curriculum online. We build on both lessons and materials from POSSE, as well as our own personal experiences.

There is a preponderance of literature written about FOSS looking at who already contributes. Surveys have been conducted to look at who is participating in FOSS [7, 8, 9]. This provides us with a look at the demographics of contributors. There is also research into the motivation of FOSS developers [10, 11, 12, 13] looking at why de-

velopers volunteer time to work on FOSS projects. Additionally there is a body of work that considers the "newbie" experiences approaching and joining FOSS projects [14, 15, 16], a great deal of which focuses on the lack of diversity within FOSS developer populations and ways to affect change [15, 17, 18]. What is lacking from this body of work is a robust foundation for creating a pedagogical approach for curriculum that supports a platform for "newbies" to become involved.

To address this issue, both instructors decided to build on the Communities of Practice (CoP) theory, though to different degrees. Efforts aimed at supporting diversity in CS classrooms often builds on, implicitly or explicitly, the CoP framework (e.g. pairs programming [19, 20] and peer mentoring [21]). Research shows that women often turn away from CS, at least in part, because they are not shown what they can do with their knowledge in real life [22]. What they lack is the introduction to the CoP that they are joining.

Wenger defines communities of practice as "groups of people who share a concern or a passion for something they do and learn how to do it better as they interact regularly" [23]. This mirrors the FOSS way of doing things, and could serve as a theoretical framework for designing effective interventions. Wenger goes on to describe "three dimensions of the relation by which practice is the source of coherence of a community" [24]. These three dimensions are:

1. Mutual Engagement – practice exists in the relationships between people, developed as they engage in practice, whose meanings are negotiated with one another. Diversity is important in this engagement as each person brings different skills and competency to the practice.
2. Joint Enterprise – the enterprise the community is engaged in is defined by negotiation. The enterprise is thus defined by participants as it develops. This gives each participant a deep feeling of ownership of the enterprise and accountability to the community.
3. Shared Repertoire – as the enterprise is negotiated shared resources are developed. This includes artifacts, and routines, words, tools, stories, symbols, actions and concepts that are negotiated over time [24].

CS courses have traditionally focused on individual achievement and competition rather than cooperation, which does not reflect industry models, more focused on group accomplishments and team work [25]. The use of techniques such as pairs programming, a cooperative model of learning and development, has been proven beneficial [19]. Technical careers often rely on teamwork, and many companies use agile methods and extreme or pair programming.

Research conducted by POSSE participants has shown that it is both possible to teach students to participate in FOSS in a classroom setting, and that students find participating in Humanitarian FOSS (HFOSS) projects especially rewarding [26]. It is important now to create a body of literature that addresses the specific approaches that lead to success, failures and lessons learned, in order to provide a robust platform from which to develop more specific curriculum.

3 Learning Objectives & Pedagogic Approach

This paper details the experiences of two instructors teaching two independent courses in FOSS development at two separate universities over the course of two years. The courses were taught using slightly different pedagogical styles, one providing more structure and guidance, as well as having a strong emphasis on in-class collaboration (Course A), and the second (Course B) following a more free-form individual-based structure. Both courses were built around the concept of building a CoP in the class, though to different degrees.

The goals of the two courses were the same: Providing students with the cultural and technical background to make a first contribution to a FOSS project, and the ability to identify future opportunities for contribution. The approach to meeting these goals was to provide a guided and structured process for navigating a joining process typically fraught with uncertainty and pitfalls. That said each instructor tailored the objectives to suit their target audience and class structure.

Course A was taught twice over two consecutive years at a small teaching university. Students in the first year course had junior and senior standing in Computer Science. A total of twenty-nine students were enrolled in the course. Students all worked on the same FOSS project and worked in teams to facilitate contribution. In the second year the course was opened up to upper class undergraduates in information systems and graduate students in management and information systems. There were a total of nine undergraduates and five students in the Masters program. Lessons were more guided and contributions were considered more broadly based on lessons learned from year one.

Course series B was taught twice over two consecutive years at a research university. The course was primarily offered to seniors and juniors in Computer Science, and followed a more traditional course model, where students picked their own projects to work on, and each student was responsible for their own coursework. There were twenty-four students in year one, and nineteen in year two.

Students were given some guidance in picking a project (more so in year 2 than year 1, when the lessons learned from the first year students were shared with the 2nd year students). This guidance was largely centered on specific projects to avoid or join, and organizational issues to look out for (active bug-tracker, up to date documentation, welcoming IRC/mailing list, etc.)

Both courses were designed to achieve the similar learning outcomes as follows:

1. Knowledge -
   (a) Give a definition of FOSS
   (b) Discuss the history of FOSS
   (c) Identify where to get answers to questions about Ubuntu. (A only)
2. Comprehension -
   (a) Explain the socio-political and technical workings of a FOSS project (B only)
   (b) Identify and summarize the workings of FOSS tools
3. Application
   (a) Map a generic path for joining an FOSS
   (b) Make use of FOSS tools (e.g., version control, bug tracker, IRC)

4. Analysis
   (a) Identify a project to contribute to (B only)
   (b) Identify potential resources and key stakeholders (B only)
   (c) Identify areas of participation using current skills (A only)
   (d) Separate research of the areas discovered to be addressed by individual group members (A only)
5. Synthesis
   (a) Assemble individual research into a plan to gain entry into an aspect of Ubuntu (A only)
   (b) Devise a plan as a group to contribute, using additional input from Ubuntu mentors (A only)
   (c) Make three contributions to a project (B only)
6. Evaluation
   (a) Explain to other groups how to complete tasks in one area of the Ubuntu project (A only)
   (b) Document contribution, and present to class (B only)

4 Course Design

4.1 Course A

Course A was designed around cooperative learning. The Ubuntu project was chosen for all students to work on to remove uncertainty and facilitate in-class collaboration. Ubuntu is a community that is known to be welcoming, has extensive documentation, and the availability of mentors. In addition to mentors, students also had access to weekly meetings of Ubuntu contributors and a prerelease Global Jam toward the end of the term. The Global Jam is an Ubuntu coordinated event uniting community members worldwide to improve the Ubuntu project. Students had several choices for how to contribute, including documentation, design, development, bug triage, and testing. Students were assigned to groups based on their area of interest. Each student had a mentor assigned to them.

As a co-operative classroom the course was taught in a lab, giving students ample hands-on experience. There were no lectures in this course; rather time was spent in discussion. If a group hit a barrier they could not overcome, they either sought help from other groups, mentors or the instructor. The term was divided with the beginning focused on history and culture, both of FOSS in general and Ubuntu specifically, followed by an introduction of tools used in FOSS, including IRC, version control, and bug-trackers. This work was used as the foundation for contribution.

After the first four-week introduction, students began to focus on contributions. As a starting point all students were assigned to bug triage, documentation, or testing. Students were required to make at least one contribution to the Ubuntu project. Groups worked to gather information from the Ubuntu documentation and their mentors to complete this assignment. Students were assessed based on participation in their team (30%), attendance (30%), and contributions to Ubuntu (40%). The final piece took into consideration the difficulty of the contribution. Fixing a bug title did

not carry as much weight as testing code or creating quick lists (a dropdown menu in Ubuntu Unity giving quick access to common tasks for applications).

4.2 Course B

Course B was designed to more closely mirror the typical lecture-based classroom experience, as well as give students more flexibility in what they wanted to get involved in. Self-motivation, the desire to scratch your own itch, is an important part of what drives FOSS contributors. This freedom of course comes at a cost; there are no guarantees that students won't pick hostile or non-productive projects, and the guidance and support that they can receive from the rest of the class and instructor are limited because each project is likely to differ in terms of tools used, customs, and availability of documentation. We chose to use this structure for this class because the student population was deemed to be more mature and experienced, and thus potentially better able to deal with potential setbacks.

Despite the divergence in projects, and that each student worked independently, there was an important social component, building on the CoP theory. There were weekly oral status reports before the whole class, and more lengthy bi-weekly experience reports and discussions. Students shared their progress, lessons learned, and pitfalls encountered, and gave each other advice on how to proceed. This helped build shared repertoire, and mutual engagement, key components to a CoP.

Lectures were based on materials from *The Cathedral and the Bazaar* [27] and *Producing Open Source Software* [28]. Most of the lecture time was aimed at providing students with the required background, including an understanding of process, and the tools used in FOSS. A lot of the technical material was front-loaded to get students up and running quickly, and the second half of the course was designed around discussions of open problems in FOSS and speakers talking about their projects.

Students were evaluated through a midterm and final exam focusing on the historical, political and licensing frameworks of FOSS (30% total grade). They had small assignments distributed through the early part of the term aimed at helping them get familiar with the organization of a particular project, and the tools used in said project (org overview and mapping of resources; use of IRC and mailing lists; and code repository and bug tracking). This accounted for 20% of their grade.

The remaining 50% was distributed over 3 contributions of the students choosing to their project of choice (bug report, documentation, testing or code patch, no more than 2 of any one). Contributions had to be accompanied by a report reflecting the importance of the work, and the process followed to create and submit it to their project. Students had bi-weekly project reports and updates to their classmates, which kept everyone up-to-date, and helped disseminate best practices and pitfalls.

Because of the short duration of a term and the long review process associated with many projects, contributions were graded by the amount of effort and whether the process was adequate rather than whether the contribution was ultimately accepted. Acceptance was nice, but not always achievable to a true novice on the first try.

5 Results

The final analysis of course A showed that in year one 25 out of 30 students (83%) made some contribution to the project. Out of those 25 students 16 (53% of the total) made more than five contributions of varying degrees of difficulty. Additionally 3 out of 5 women and 4 of 6 non-white were part of the groups thathad the highest performance levels. Analysis of year two shows that 11 out of 14 students (79%) made contributions to the project and of those 11 students 5 (45%, of the total) made more than five contributions. Given the difference in skill sets due to the mix of CS, IS and MIS, the level of contribution was not considered in year two.

Course B similarly had a high degree of success, with 19 of the 24 students completing their three contributions in a satisfactory manner in year one (79.2% of students), and 17 of 19 students (89.5% of students) in the second year. Of those who did not complete all three assignments, all completed at least one contribution in year one, and two in year two. For the second year, student evaluations indicated that this was one of the most worthwhile courses the students had completed, and more than half the students indicated that they intended to continue working on the project they had chosen. For the first year, evaluations were mixed, primarily due to frustration with the projects they had selected and issues of non-responsiveness and lacking or misleading documentation, and fewer than a quarter of students expressed an interest in continuing their work on the projects after the end of the class.

6 Discussion and Lessons Learned

While it is difficult to objectively evaluate the effectiveness of the courses we designed, other than in terms of meeting their learning objectives, we judge them to have been a success. Joining a FOSS project for the first time is a very time-consuming and uncertain process, and for most students, either course helped them successfully navigate the learning process in a very limited amount of time.

As should be evident from the description of the courses presented, a significant amount of the learning was still self-directed; the students had to figure out what to do for themselves. What the course did provide was a social and technical context for doing so. The class gave them a goal, helped explain some of the organizational and cultural issues they were previously unfamiliar with, and the understanding that this process is difficult, and that it is OK to be lost. With this context, the students were largely able to navigate the process with some minimal support and encouragement.

Though we only have anecdotal evidence to support this, we found that the social context, the CoP framework that we sought to build, was immensely valuable. While many students initially were reluctant to admit problems or failures, and indeed would likely have struggled in silence with these until they succeeded or gave up, by the midpoint of the course there was a definitive feeling of camaraderie among the students. They shared their frustrations with each other openly, and were able to offer advice and coaching to each other unprompted. Forming in-class teams or not does not seem to have been a factor for success or failure.

That said, we did experience some significant problems in our courses. The openness of course A (make any number of contributions of any kind) coupled with the size of the Ubuntu project turned out to be a significant obstacle. As a result students reported being overwhelmed and unable to find a place to begin. It is worth noting that students who interacted with their mentors and each other on a regular basis found it easier to contribute to the project. It became apparent that the course needed more structure and direction. Students also needed to be taught how to use their mentors to gather information. Just providing mentorship did not ensure that students knew what to ask or how to get started talking to their mentors.

Course B was in some ways more open than Course A, but students were directed to either smaller projects, projects they already had experience with, or projects that had an established mentor network. The expectations in terms of contributions were better defined however. Fewer students reported being overwhelmed as a result.

Because of the openness of course A, evaluation was difficult. Grades were based on participation, contribution, and attendance. The difficulty evaluating students stemmed from not having clear guidelines and expectations. This, coupled with the lack of structure and overwhelming size of the project, pointed to a solution that would address all three problems. Using a smaller project written in a language students had experience with would provide students with a more moderate learning curve. The course needed assignments to provide the means to gain access to the project, but also as a means to evaluate students' work. Course B on the other hand turned out to be much easier to evaluate, and the exams and minor assignments helped students feel less worried about their grades.

The end result of using Ubuntu was the realization that the project was too large. Although there were many places to participate, and the community was very welcoming, most of the students reported being overwhelmed with the documentation of the project. While the documentation was exhaustive, the sheer volume made it difficult for students to find answers to their questions and a place to start. Fifteen out of twenty-seven students listed being overwhelmed by the amount of documentation and/or finding an entry point into the project because it was so large.

Not picking a project for students however led to more work for the instructor, and some additional uncertainty among students early in the term, but in the second year we developed guidelines for selecting better projects, and students were more devoted to their tasks.

These courses will continue at both universities, and the curriculum is available for others seeking to adopt or design their own curriculum <links to be shared after review>.

7 Conclusions

Using FOSS in higher education serves both the students – by providing real world experience, as well as the FOSS community – by growing a pool of potential developers. Courses that emphasize hands-on experience and the completion of real contributions to real FOSS projects were able to achieve a very high success rate.

Following a CoP model, whether strictly or loosely interpreted, contributed to this success, and helped students overcome the confusion and frustration of dealing with an often unstructured learning challenge.

8 References

1. Deshpande, A., Riehle, D.: The total growth of open source. Open Source Dev. Communities Qual. 197–209 (2008).
2. Shibuya, B., Tamai, T.: Understanding the process of participating in open source communities. Emerging Trends in Free/Libre/Open Source Software Research and Development, 2009. FLOSS'09. ICSE Workshop on. pp. 1–6 (2009).
3. Trapasso, E., Vujanic, A.: Accenture Newsroom: Investment in Open Source Software Set to Rise, Accenture Survey Finds. (2010).
4. Hansson, D.H.: Reduce the risk, hire from open source, http://www.loudthinking.com/arc/000505.html, (2005).
5. Ellis, H.J., Chua, M., Hislop, G.W., Purcell, M., Dziallas, S.: Towards a Model of Faculty Development for FOSS in Education.
6. Teaching Open Source, http://teachingopensource.org/index.php/Main_Page.
7. Ghosh, R.A., Glott, R., Krieger, B., Robles, G.: Free/libre and open source software: Survey and study. Maastricht Economic Research Institute on Innovation and Technology, University of Maastricht, The Netherlands, June (2002).
8. David, P.A., Waterman, A., Arora, S.: FLOSS-US: the free/libre/open source software survey for 2003. Stanf. Inst. Econ. Policy Res. Httpwww Stanf. Edugroupfloss-Us Accessed. 20, (2004).
9. Lakhani, K., Wolf, B., Bates, J., DiBona, C.: The boston consulting group hacker survey. Boston Boston Consult. Group. (2002).
10. Hars, A., Ou, S.: Working for free? Motivations of participating in open source projects. System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on. p. 9–pp (2001).
11. Hertel, G., Niedner, S., Herrmann, S.: Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. Res. Policy. 32, 1159–1177 (2003).
12. Roberts, J.A., Hann, I., Slaughter, S.A.: Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the apache projects. Manag. Sci. 52, 984 (2006).
13. Ye, Y., Kishida, K.: Toward an understanding of the motivation of open source software developers. Software Engineering, 2003. Proceedings. 25th International Conference on. pp. 419–429 (2003).
14. King, S., Kuechler, V., Jensen, C.: Joining Free/Open Source Software Communities An Analysis of Newbies' First Interactions on Project Mailing Lists. Presented at the 2011 44th Hawaii International Conference on System Sciences January 1 (2011).
15. Kuechler, V., Gilbertson, C., Jensen, C.: Gender Differences in Early Free and Open Source Software Joining Process. Open Source Syst. Long-Term Sustain. 78–93 (2012).

16. Park, Y.: Supporting the learning process of open source novices: an evaluation of code and project history visualization tools. (2008).
17. Byfield, B.: Sexism: Open Source Software's Dirty Little Secret - Datamation, http://www.datamation.com/osrc/article.php/3838186/Sexism-Open-Source-Softwares-Dirty-Little-Secret.htm.
18. Levesque, M., Wilson, G.: Women in software: Open source, cold shoulder. Softw. Dev. URL Consult. 20 Febr. 2005 Httpwww Sdmagazine Comdocumentss. 9411, (2004).
19. McDowell, C., Werner, L., Bullock, H.E., Fernald, J.: Pair programming improves student retention, confidence, and program quality. Commun. ACM. 49, 90–95 (2006).
20. Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., Balik, S.: Improving the CS1 experience with pair programming. ACM SIGCSE Bulletin. pp. 359–362 (2003).
21. Cohoon, J.M.G., Gonsoulin, M., Layman, J.: Mentoring computer science undergraduates. Hum. Perspect. Internet Soc. Cult. Psychol. Gend. 4, 199–208 (2004).
22. Margolis, J., Fisher, A.: Unlocking the clubhouse. MIT Press (2002).
23. Wenger, E.: Communities of practice: A brief introduction. Retrieved Oct. 1, 2008 (2006).
24. Wenger, E.: Communities of practice: Learning, meaning, and identity. Cambridge Univ Pr (1998).
25. Howell, K.: The experience of women in undergraduate computer science: what does the research say? ACM SIGCSE Bull. 25, 1–8 (1993).
26. Morelli, R., Tucker, A., Danner, N., De Lanerolle, T.R., Ellis, H.J., Izmirli, O., Krizanc, D., Parker, G.: Revitalizing computing education through free and open source software for humanity. Commun. ACM. 52, 67–75 (2009).
27. Raymond, E.S.: The cathedral and the bazaar: musings on Linux and open source by an accidental revolutionary. O'Reilly & Associates, Inc. (2001).
28. Producing Open Source Software, http://producingoss.com/.