A Course in Open Source Development

Timothy A Budd
School of EECS
Oregon State University
+1 737 5581
budd@eecs.oregonstate.edu

ABSTRACT
A course in Open Source Development can be greatly beneficial to the student. Yet such courses are not commonly found in the college curriculum. This paper describes a course in open source development that has been successfully and repeatedly offered at Oregon State University.

Categories and Subject Descriptors
K.3.1 [**Computer Uses in Education**], K.3.2 **[Computer and Information Science Education]**

General Terms
Management, Performance, Design.

Keywords
open source, curriculum, education.

1. INTRODUCTION
I believe that a course in open source development can be one of the most important classes in the undergraduate curriculum. The reason for my belief can be summed up in three words: *code*, *community*, and *tools*. Nor am I alone in my opinion. No less a notable than Dave Patterson, Professor at UC Berkeley, and past president of the ACM, said in an open letter to the community as he was leaving his term at the ACM that a course in open source development was the number one "course I would love to take" as part of the computer science education in the 21st Century [1].

2. Code, Community and Tools
2.1 Code
Students don't get enough experience in reading large code bases. To illustrate this point, think about teaching programming in comparison to teaching writing, or teaching a foreign language. Young children are taught to read before they write, and are expected to have read a considerable amount before they are expected to produce any serious writing (such as an essay). It is only when we teach computer programming that we expect students to write before they have read any considerable body of work. Twenty years ago this attitude was understandable, as most commercial software development organizations treated source code as intellectual property or trade secrets. This attitude has been described by Dick Gabriel [2]:

"The effect of ownership imperatives has caused there to be no body of software as literature. It is as if all writers had their own private companies and only people in the Melville Company could read *Moby-Dick*, and only those in Hemmingway's could read *The Sun Also Rises*. Can you imagine developing a rich literature under these circumstances? Under such conditions, there could be neither a curriculum in literature nor a way of teaching writing. And we expect people to learn to program in this [fashion]?"

With the advent of open source there is little excuse for not exposing students to large code bases. How is a student going to know what is good, what is bad, what works and what doesn't, unless they have seen a lot of examples of all different styles? So if nothing else, open source belongs in the curriculum because of the code.

2.2 Community
But it is much more than the code. Open source projects require students to interact with a *community* of developers, spread geographically, temporally, culturally, attitudinally, and from different backgrounds. Again contrast this with the typical university curriculum, where even if students work in teams they are small (often no more than four students) and uniform (all students in the same course, or at the same point in their development, such as in senior project courses). Experience working in teams is almost always cited by our industry advisory panels as one of the major experiences our students should have prior to graduation. Working on an open source development project provides the student with an experience that is almost impossible to duplicate in a classroom only environment.

Open source development projects can also allow the student to interact with socially relevant community organizations. For example, many of our students at OSU have been involved in the United-Nations sponsored One-Laptop Per Child project [3]. Two of our students, sophomores at the time, ported the word processor that is now being included in the core distribution for this system. This means that their work could potentially be used in the next few years by millions of children in the third word. Socially relevant projects such as the OLPC can be extremely attractive to many students.

2.3 Tools
But it is not just the code or the community, it's the *skills*. One of the most important reasons why I think experience in open source

development should be part of every university curriculum in computer science is because you can never predict what tools or skills will be required for a given open source project. Every project is different. So the student is forced to become an autodidact, a self-teacher and self-learner. This is frequently a tremendously empowering experience for the student. Again, contrast this with the typical course experience, where everything that needs to be taught is covered in the textbook, or in lectures. In an open source development course, students are forced to figure out what it is they need to know, how to find information about this topic, and how to teach themselves.

If you stop to think about it; suppose there was only one topic we could teach in the university–shouldn't the ability to teach oneself be that topic? So helping students to become self-teachers is perhaps the most important reason for trying to get students involved in open source development projects.

3. Why aren't there more courses in Open Source Development?

Given all the benefits to having students work on open source development projects, why are there so relatively few courses at the university level that teach this topic? I think there are two major reasons.

First, professors teach what they have been taught, have first-hand experience with or, barring that, what they can learn from textbooks. When it comes to open source development, there are just too few individuals that fall into the first two groups, and (as far as I am aware) no examples of the last. (There are lots of trade books, but no textbooks). Most professors feel uncomfortable being cast as a mentor for a process they don't understand themselves.

Second, the process is not been clearly articulated. How do students become involved in open source projects? Most fall into it through the back door, because they become involved in a project that satisfies a personal passion. They then stumble around, or if they are lucky find a mentor who will help them, and along the way pick up whatever skills they need. But, again as far as I know, nobody (including myself) has described in general terms the process or a framework for becoming involved in open source projects.

3.1 Open Source Merit Badges

A few years ago, some friends and I were talking about this problem, and came up with at least a catchy title and metaphor, if not much else. The title was "Open Source Merit Badges". The idea was that we needed a series of tasks that would satisfy a number of requirements.

- Each task would allow the student to develop an essential skill
- Each task would have an identifiable or measurable outcome, so that it can be easily determined whether or not the task has been performed
- The tasks would be in a sequence of increasing difficulty. The first ones should be easy enough that they would not be intimidating, and each step should build on the previous, so that they, too, would not be intimidating. The achievement of each task would give the student a measurable indication of progress
- Each task would be clearly articulated so that the student would understand the job to be done, but open ended enough that the student would have to explore on their own in order to achieve the task. This is part of the process of forcing the student to become a self-learner.
- The final steps in this process would leave the student in a position to be a useful contributor to the ultimate task at hand, e.g., working as an open source developer.

I would like to say that as part of the course I will shortly describe I have solved this problem, but alas it is not the case. However, it does provide a useful structure for asking questions. As part of trying to develop this framework I've started asking people all sorts of questions such as "how did you get involved in project zzz", or "what skills do you think students need to work on project zzz?"

4. A course in Open Source Development at Oregon State University

The remainder of this document will describe features of the course on open source development I have successfully taught for two years in the department of computer science at Oregon State University. The course changes each time it is offered, and is still rapidly evolving, but the following basic features have been tested in a classroom environment and been found to work.

The course is a combination of lectures, discussion, outside speakers, and student participation in an actual open source development project.

The starting point for almost any explanation of open source must be Eric Raymond’s essay on *The Cathedral and the Bazaar* [4]. Although this notable essay is starting to be somewhat dated, it is written in an accessible style, readily available, and an excellent taking off point for in-class discussions. I usually have this discussion take place in the 2nd or 3rd class meeting, after a general lecture that introduces the basic tenets of the open source movement.

The idea of *community* is central to everything about open source. It is important to make the course itself feel like a community. One way I do this is to place all course related material on a wiki that all members of the course can, and are expected to, contribute and edit. Of course some pages (the class calendar, syllabus) must be password protected. But students must write their own pages on which they introduce themselves, as well as helping to build a bibliography of papers, projects, and other on-line source material. In addition, every class has an associated electronic mailing list, and students are encouraged to correspond with each other in the e-mail list.

One of the first assignments is to make a contribution to Wikipedia, one of the largest and most active open source projects in existence [5]. This assignment helps students to get over the hurdle of participation, even if a large number of submissions eventually end up being edited or removed. Students are given a week to find a topic and make their submission to wikipedia. This is followed by an in-class discussion on the assignment, and on the process by which a large number of people can together create useful artifacts.

Another important early assignment is to have the student research an open source project of their own choosing. This is a necessary step they need to take prior to participating in a project, but the possibility of discovering how a project works by lurking anonymously around the web pages somehow makes students feel more comfortable. The student must discover, and describe, not only the topic of the community, the features of the application, but also how the community around the application is organized, such as how they communicate bugs (bug tracking software, email forums) and communicate with each other (IRC channels, email forums, bug days). This will often expose the students to technologies they might not have seen previously, such as mail forums, IRC channels, bug tracking lists, and so on.

Outside Speakers provide an excellent way for students to learn about open source from a variety of different perspectives. We are blessed in Oregon with having a number of world-class speakers near at hand, but almost any large institution will undoubtedly have some connection to the open source world. If you ask around, you will probably discover that your library, your university IT departments, or other internal groups are making significant use of open source technology.

While outside speakers are good for providing alternative points of view, it is also important for students themselves to both have experience in public speaking, and to learn from each other. I do this using *Tech Talks*. Tech talks are student presentations, typically fifteen to twenty minutes in length, which cover a course related topic. I allow three different categories of talks. The first is a presentation of a tool or technique, such as discussing CVS, or git, or quilt, or bugzilla, or any number of countless alternatives. The second type of talk is a community introduction. This is actually an extension of the work they performed in their first investigation of an open source project, but this time presented in a public presentation, rather than as a report. The third type of talk combines the community presentation with a discussion of how a particular patch was found and fixed, and the submission process used by the project. Again, this can easily duplicate material submitted for the paper or patch assignment (below), but the public presentation in a class setting from a peer student can be very effective, particularly in the early portions of the course.

Giving tech talks is the most common way for students to gain *participation points*. I know that students can learn from each other as much as they learn from me or any of the other speakers. So other ways that students can earn participation points is by providing help sessions outside of class, such as on weekends or evenings. In order to gain participation points students must open these help sessions to all students, advertise (such as by an announcement in the class mail list), and afterwards describe the outcome (by a short email to the professor). Other means of achieving participation points can be determined on a case-by-case basis.

At the heart of the course is a requirement that every two weeks students must submit a *paper or patch*. That is, either select two or more papers from the available literature and write their own document summarizing and contrasting them (a more or less traditional "paper" assignment), or submit a patch to an existing open source project. The contribution of the latter need not be large, and indeed most patches are very simple. The word "patch" is a shorthand, as the submission need not be code, but could be in the form of documentation or other artifacts. The key idea is that the student is not exploring the product as much as the process. In addition to submitting the patch, the student must write a paper in which they describe the project, the community, and the process of not only finding and fixing the bug, but submitting the artifact, and the process the community will following in vetting or approving the submission.

Students are encouraged to work together on portions of the patch process. Students with common interest can find each other, or be paired, while they explore how a particular open source community operates. While the patch that they eventually submit must be individual, the exploration of the code base, and the investigation of the community aspects of a project can be performed with the help of other students.

Peer review is another key concept in open source. All papers are submitted in a public fashion, either by placing them on the wiki, or by placing them in a web accessible directory (all university students have a public web directory) and e-mailing a short summary and link to the class on the course web page. For each of the four paper or patch assignments each student is assigned one other member of the class to review. In addition to their assigned review, the student selects two other students to review. The reviews of the paper or patch assignment comment on both the quality and the significance of the submission.

I use a common +-?DA format for reviews: the plus indicating what was good about the submission, the minus indicating what could stand improvement, and question for what was difficult to understand from the paper, and the DA (devil's advocate) for proposing contrary views or possibilities. The peer reviews can be very short (often only a sentence or two in each section) but have the effect of forcing the students to look at what other people are doing. The peer reviews are also placed in a public location, typically by posting to the course e-mail list. (Learning to deal with a vast amount of e-mail is also an important part of dealing with most open source projects).

4.1 Grading and Evaluation

As I was developing this course I thought for a long time how to do examinations and evaluation. In the end, I decided that these were almost impossible. Certainly I could test vocabulary (e.g., define the terms "open source"), but beyond that there was very little that could be tested in a traditional fashion. Similarly even grading the paper or patch assignments was problematic, since they were each very individualized. After a few students tried to test the limits of my acceptance I did develop a minimal rubric, chiefly describing features such as appearance, length, and appropriate topic. But with just a small amount of attention to detail most students can meet the stated objectives.

For the most part the assignments are graded on a basis of either they did it or they didn't do it. The current syllabus for the course states that submission of all the required assignments except for the tech talk presentations is sufficient for a B grade. To obtain a higher grade students must not only submit all the assigned paper work, but also gain some participation points through giving a tech talk or helping other students.

4.2 Students with Differing Backgrounds

Going into the course students will undoubtedly have a very diverse background. A few students may already have extensive experience with Linux and possibly even with open source development. Many students will not. The upper third of the class will be successful and gain valuable experience almost without any assistance from the instructor. It is the students who are not in the upper third that must be given the most attention. These students must not be allowed to become discouraged, thinking that they cannot possible succeed because they do not have the same background as the leaders. Encouragement must be constantly provided, and the mantra that it is the *process* that it important, not the *product*, must be frequently repeated. That is, it is not the significance of a patch that is important, but that the student learns about the process of contributing to open source projects. I fully believe that every student has the potential for success, and the fact that they can teach themselves new topics is often an eye opening experience.

5. Conclusions

A course dealing with open source development can be a great benefit to the undergraduate student. Despite this, such courses are relatively uncommon in the undergraduate curriculum. A likely reason for this is the fact that few instructors have first-hand experience with open source. In this paper we have described one course that has been successfully offered at Oregon State University several times. This course can serve as a model for similar courses presented elsewhere.

6. REFERENCES

[1] David A. Patterson, "Computer Science Education in the 21st Century," *Communications of the ACM*, Vol 49, Number 1, March 2006.

[2] Richard P. Gabriel and Ron Goldman. *Mob software*. Oct 2000. Presented at the ACM Conference on Object-Oriented Systems, Languages and Applications on Oct 19, 2000. http://www.dreamsongs.com/MobSoftware.html

[3] OLPC, for details see http://laptop.org

[4] Eric S. Raymond, *The Cathedral & and Bazaar, Musing on Linux and Open Source by an Accidental Revolutionary*, O'Reilly & Associates, Sebastopol, CA, 1999.

[5] www.wikipedia.org