Challenges and Opportunities in an Open Source Software Development Course

Cay S. Horstmann
San Jose State University
Cay.Horstmann@sjsu.edu

ABSTRACT

I report on the design of a course in open source software development that has been offered as a semester-long class to computer science students at San Jose State University as well as an international summer school with participants from HEIG-VD (Yverdon, Switzerland), SJSU, CSU Long Beach, and Arizona State University.

The theoretical part of the course covers the foundational underpinnings through reading and analysis of key white papers, software licenses, documented development practices, and case studies of several important projects of different degrees of complexity. In the practical part, students learn to use the "tools of the trade", in particular, source control, build automation, and patching. Students study open source projects, identify and implement fixes and improvements, interact with project developers and committers, and aim to have their contributions accepted.

Categories and Subject Descriptors

K.3.2 Computer and Information Science Education

General Terms

Design, Economics, Experimentation, Legal Aspects

Keywords

Open source software, software engineering education

1. INTRODUCTION

In recent years, open source development has become an important force in the software engineering discipline. Software engineers are increasingly expected to be able to evaluate open source solutions, integrate with open source products or libraries, and contribute fixes and enhancements to open source projects. However, the traditional university curriculum offers little guidance for these activities.

Moreover, certain aspects of open source development have pedagogical benefit in a computer science or software engineering program. The study of open source projects exposes students to large bodies of code and tools for build automation, version control, and project management. Most importantly, students experience the social dynamics of working on a large project. These are valuable skills for computer science students to develop—lack of tool knowledge and insufficient team work experience are the two most common complaints from our department's industry advisory board.

In [5], Patterson describes a "Course I would love to take: Join the open source movement". That hypothetical course tasks students with providing documentation, debugging, and implementations of new features in open source projects. Patterson suggests that it is inspiring for computer science students to work on real production projects—an opportunity that civil engineering or history students do not have.

[4] describes a course in which students contribute programming problems to an open-source assessment system. The focus is on teaching of software quality assurance, not open source development. As described in [1], Rice University students have participated in the development of the open source Dr. Java programming environment. The authors cite this activity as valuable for narrowing "the gap between writing toy programs and developing reliable software products". Participation in a humanitarian open source project by students in a liberal arts college is described in [2]. One motivation is to attract a larger group of students to computer science by demonstrating that the profession can make meaningful contributions to society. Wayne State University offers a course in software evolution that studies open source projects [6].

Unlike [4, 1, 2], which use a single open source package, or [6], which focuses on a single aspect of open source development, the course that I developed at SJSU attempts to give an overview of the entire open-source development process..

2. COURSE DESIGN

The course was designed as a semester-long class for senior undergraduate or graduate students. (The graduate students were required to work on more challenging projects, and they had to produce a research paper.) A course in software engineering was a prerequisite.

Expected learning outcomes are:

- Explain the core principles of the open source development model
- Explain the ramifications of choosing among open source licenses
- Explain the key differences between open source and commercial development styles
- Describe the features, scope, goals, and implementations of key open source projects
- Effectively use the tools that are commonly used in the open source community for source control, cross-platform development, build automation, patching, testing, documentation, and communication
- Analyze and modify open source code bases
- Participate in open source projects and contribute fixes and improvements

Lectures, labs, and readings cover these topics:

- General principles of the open source movement (10%)
- The business of open source (10%)
- Legal issues: licenses, copyright, patents (10%)
- Organization of open source teams (10%)
- Open standards and the standardization process (5%)
- Open source tools (35%)
- Architectural choices in open source projects (20%)

Students gain experience with building and inspecting the code base of open source projects, ranging from a small Java-based project to a massive one such as OpenJDK or Mozilla Thunderbird.

Each student has to pick an open source project, engage in discussions with the developers, solve and contribute a task of their choice, and aim to have the contribution accepted.

When I taught the course in Fall 2007, successful student contributions were mostly simple bug fixes, but several students carried out impressive work, diving deeply into complex code or redesigning autotools scripts.

I taught an abbreviated version of the course at an international summer school with participants from HEIG-VD (Yverdon, Switzerland), SJSU, CSU Long Beach, and Arizona State University. Because the students were very motivated, I was able to cover most of the technical material in two weeks. Of course, there was no expectation for students to engage with an actual project in that time frame.

3. OBSERVATIONS

3.1. Challenges

When offering a course on open source development as an elective course that is open to all students, one must be prepared for a wide variation in student skills. Some students with work experience are used to working with a small part of a large code base. (About 50% of our students work at least part-time in a computing-related job, but not always as software developers.) However, most students have never encountered a large code base and find it very intimidating.

The students without prior work experience tended to have an unrealistic expectation of code quality. They were surprised that the code that they encountered in real projects was not as polished as textbook code, and that it suffered from many of the shortcomings of their own code such as lack of documentation, clumsily implemented algorithms, and inconsistent formatting.

Many senior undergraduate students are unfamiliar with the "tools of the trade" such as version control or build automation, even if they have completed a course in software engineering (which is a prerequisite). Unfortunately, there appears to be a wide variation in the delivery of a software engineering course. All students knew the definition of version control or build automation, but only a few had actually been introduced to actual tools in their course.

About 2/3 of the students in the course had never worked on any platform other than Windows XP. Cross-platform issues were foreign to them. As a result, they made bad assumptions about path names, library locations, character encodings, and so on.

Apparently, only a small minority of computer science students has any familiarity with the world of business or the law. I was surprised how few of them followed computer industry news, even though it would obviously be in their self-interest to do so. This lack of basic background knowledge makes it difficult to have meaningful discussions on business motivations or legal issues. When designing the course, I provided an option for students to obtain a large amount of work credit on business and legal issues, thus enabling students with inferior programming skills to succeed. Sadly, those students were also most lacking in the ability to reason about business, social, and legal issues.

Finally, and perhaps most importantly, interacting with developers of an open-source project requires a certain amount of social skills and self-confidence. Some students found it difficult to ask meaningful questions, or to follow the social protocol that is necessary to establish credentials.

3.2. Curricular Benefits

Overcoming, or at least squarely facing, each of challenges listed in the preceding section can yield substantial curricular benefits in a program whose principal objective is to train practitioners. (Only a small number of our students pursue a Ph.D.)

When students become successful working with a large code body, by running builds and making small modifications, they lose fear and their confidence level increases tremendously. This was the most positive experience cited by students in course evaluations.

Some students with poor coding habits were motivated to do better when they saw similar bad examples in "real" code, whereas they had been resistant to change when they were exposed to clean textbook code. I suspect that learning from negative examples can be quite powerful. A student who had to suffer through poorly documented or convoluted code might be motivated not to inflict similar suffering on others.

While it is unfortunate that so few students knew the "tools of the trade" when they entered the class, it offered a perfect teaching opportunity. Version control and build automation tools quickly became second nature. Students also gained respect for configuration and platform issues since it clearly isn't enough that "it runs on my machine" when you work with collaborators around the globe.

For many students, this was the first course that required them to consider how their profession enables companies to make money, and what legal framework it operates in. Students presented papers from [6] that discussed business factors for using and contributing to open source software, team structure, and legal issues. They were also tasked with following specific industry news such as the scandalous OOXML standardization that was unfolding as the class progressed.

I had expected that in a world of Facebook, students would be naturally extroverted in an online setting and be willing to ask questions and make contributions that are there for all the world to see. That was, of course, naive. We spent a good amount of time practicing how to ask good questions, as described in [7]. Much of this is common sense: pare down problems to the basics, pay attention to detail, show that you did your homework, supply essential configuration information. We practiced supplying patches and describing their utility. We discussed the social structure of open source projects and the motivations of the stakeholders. These were excellent lessons for the students who had given little

thought to the difference in social norms between students and professionals.

4. CONCLUSIONS

A course in open source development can impart skills that are rated as useful by industrial employers, in particular

- working with large code bases
- use of the "tools of the trade"
- working in distributed teams

It allows students to think about the business and legal environments of their future professional work.

For such a course to be successful, it is important to have realistic expectations of student capabilities. Sufficient time should be spent on tool usage and the social aspects of collaboration, preferably at the beginning of the course, so that students can put this knowledge to use when interacting with project collaborators. With this preparation, it is feasible for senior undergraduate students to make minor contributions to actual projects in a semester.

Ideally, essential software tools and the use of larger code bases should be taught much earlier in the curriculum. For example, it would be a good idea in junior level courses to use version control for homework submissions, and to ask students to extend existing code rather than writing everything from scratch. It would also be desirable if students had prior exposure to the business and social environment of their chosen profession, such as discussion of industry news as a part of their coursework. Hopefully, an open source development course can lead to such curricular improvements.

REFERENCES

[1] Allen, E., Cartwright, R., and Reis, C. 2003. Production programming in the classroom. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education* (Reno, Navada, USA, February 19 - 23, 2003). SIGCSE '03. ACM, New York, NY, 89-93. DOI= http://doi.acm.org/10.1145/611892.611940

[2] 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?. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education* (Covington, Kentucky, USA, March 07 - 11, 2007). SIGCSE '07. ACM, New York, NY, 551-555

[3] Feller, J., Fitzgerald, B., Hissam, S., and Lakhani, K. Perspectives on Free and Open Source Software, *MIT Press*, Cambridge, MA, 2005

[4] Gotel, O., Scharff, C., and Wildenberg, A. 2008. Teaching software quality assurance by encouraging student contributions to an open source web-based system for the assessment of programming assignments. In *Proceedings of the 13th Annual Conference on innovation and Technology in Computer Science Education* (Madrid, Spain, June 30 - July 02, 2008). ITiCSE '08. ACM, New York, NY, 214-218

[5] Patterson, D. A. Computer Science Education in the 21st Century, *Communications of the ACM* Vol. 49 No. 3 (March 2006), 27-30

[6] Petrenko, M.; Poshyvattyk, D.; Rajlich, V.; Buchta, J. Teaching Software Evolution in Open Source, *Computer* , vol.40, no.11 (Nov. 2007), 25-31

[7] Raymond, E. How To Ask Questions The Smart Way, Revision 3.6, March 19, 2008, http://www.catb.org/~esr/faqs/smart-questions.html