Chapter 38
Bridging the Academia–Industry Gap in Software Engineering: A Client–Oriented Open Source Software Projects Course

Bonnie K. MacKellar
St. John's University, USA

Mihaela Sabin
University of New Hampshire, USA

Allen B. Tucker
Bowdoin College, USA

ABSTRACT

Too often, computer science programs offer a software engineering course that emphasizes concepts, principles, and practical techniques, but fails to engage students in real-world software experiences. The authors have developed an approach to teaching undergraduate software engineering courses that integrates client-oriented project development and open source development practice. They call this approach the Client-Oriented Open Source Software (CO-FOSS) model. The advantages of this approach are that students are involved directly with a client, nonprofits gain a useful software application, and the project is available as open source for other students or organizations to extend and adapt. This chapter describes the motivation, elaborates the approach, and presents the results in substantial detail. The process is agile and the development framework is transferrable to other one-semester software engineering courses in a wide range of institutions.

MOTIVATION

Most computer science programs offer a software engineering course and view it as a critical link in ensuring the career-readiness of computer science graduates. However, too often this course is taught in terms of abstract principles, failing to engage students in real-world software experiences. Many of the skills required in industry are best learned by hands-on practice, such as the need for effective

DOI: 10.4018/978-1-4666-7363-2.ch038

communication among developers, or the need to interact with a non-technical client. Thus, students who have never engaged in a hands-on project in software engineering enter the workforce with gaps in their skills.

It is, however, difficult to bring a significant software development experience into the confines of a one-semester course in academia. The most common approach has been to introduce a "toy project," which is a small project designed by the instructor, and have students work in teams to complete the project by the end of the semester. The advantage of this approach is that students will ideally learn to work in teams and share responsibility for developing a codebase. The disadvantages are that the project may be oversimplified, and students gain no experience interacting with clients or with code written by others.

Another approach is to work with local companies in the private sector who sponsor *proprietary client-oriented software projects*. This has been used successfully by a number of schools, especially larger programs that already have established linkages with companies (Judith, Bair, & Börstler, 2003; Tadayon, 2004; Tan & Jones, 2008). Another setting that favors this approach is an internship course with the projects being developed onsite at local companies. The advantage is that students gain experience with real clients with high stakes in real projects. However, these projects are often standalone, one-off projects since companies may be reluctant to have students work on their internal codebase, or to develop mission critical software. This means that it may be difficult to get enough time and attention from personnel at the company while the students work on the project. Also, the project will normally become the property of the company, meaning that it cannot be freely shared with other schools trying to adopt a similar approach.

A third approach is to engage students in Free and Open Source Software (FOSS) development by having them contribute to *a large and active open source project*, such as Linux or Mozilla (Marmorstein, 2011; Ellis, Morelli, DeLanerolle, & Hislop, 2007). The advantage of this approach is that instructors and students can gain from the mentoring achieved through communication with the project's professional developers, and in some cases they contribute marginally to the "live" code base or the user documentation. The disadvantages of this approach are that most ongoing projects are large and complex, their developers may not be accessible, and given the time it takes to come up to speed in the project, students may gain little practical experience in a one-semester course.

A fourth approach, which occupies a middle ground between the proprietary client-oriented project model and the full-scale FOSS project, is to engage students in FOSS development via a relatively small project that fits in a one semester course, with a local nonprofit organization as the client. Local nonprofits are often happy to collaborate on these projects since they may have needs for mission-critical software systems that are not well met by the commercial software industry, yet they have limited technology budgets. Thus, it is relatively easy for an instructor to locate and collaborate with a local nonprofit. However, many instructors may still be unsure of how to get started or how to organize such a course.

This chapter describes our collective experience with the fourth approach, which we call *client-oriented free and open source software development* (CO-FOSS). The big advantage of treating client-oriented open source projects is the very openness of the project. An open source project developed in the context of one course for one client can be reused, extended, and adapted for new clients by subsequent iterations of the same course, or even by courses at different institutions. By providing not just the codebase but the course organization itself as an open source project, a collection of such projects can be built up to be used as models at different institutions. In addition, the tools and practices of open source projects provide a readymade infrastructure for software project courses.