AC 2009-1516: SOFTWARE PROJECTS USING FREE AND OPEN-SOURCE SOFTWARE: OPPORTUNITIES, CHALLENGES, AND LESSONS LEARNED

Clifton Kussmaul, Muhlenberg College

Clif Kussmaul is Associate Professor of Computer Science at Muhlenberg College and Chief Technology Officer for Elegance Technologies, Inc., which develops software products and provides software development services. Previously he worked at NeST Technologies, and Moravian College. He has a PhD in Computer Science from the University of California, Davis, master's degrees from Dartmouth College, and bachelor's degrees from Swarthmore College. His professional interests and activities include software engineering, entrepreneurship, digital signal processing, cognitive neuroscience, and music.

Software Projects Using Free and Open Source Software: Opportunities, Challenges, & Lessons Learned

Abstract

Software projects play a major role in software engineering (SE) education, have a long history and extensive literature, and present instructors with a variety of pedagogical options. For example, if students build an entire system, they see more of the early development stages and may have more choices, but the scope is limited and they may have to make critical decisions based on insufficient experience. If students extend or enhance an existing system (often from previous terms), they experience larger systems, but see less of the early development stages.

Free and Open Source Software (FOSS) can provide useful alternatives. FOSS generally refers to software that is distributed without charge and with the original source code, so that anyone can fix defects, add enhancements, or otherwise modify the software and share their changes with others. FOSS can provide SE projects with several benefits. First, successful FOSS usually has a robust implementation, including high cohesion, low coupling, and effective tests and documentation, since many developers work on them briefly or intermittently. Second, FOSS usually has a varied user community, which demonstrates the role and value of communication and supporting tools, such as discussion forums, version control, and task or defect tracking systems. Third, students may already be familiar with FOSS as users.

Faculty can help students by using a five step "USABL" model in which students use FOSS projects, study the project as a worked example, add minor enhancements, build larger components, and finally leverage FOSS for other purposes. This paper describes experiences using FOSS and this approach across a computer science (CS) curriculum and particularly in a sophomore-level SE course and in capstone software projects. First, it briefly reviews SE course and project design, and FOSS. Second, it describes the five step model, and a series of related activities, assignments, and projects. Third, it concludes with benefits and future directions.

1. Introduction

Software projects play a major role in software engineering (SE) education, and have a long history and extensive literature[7]. General principles for instructional design can help instructors to design more effective projects and project-based courses. These principles suggest that Free and Open Source Software (FOSS) can be used to support and enhance SE projects and project courses. This paper proposes that faculty can help students by using a five step "USABL" model in which students *use* FOSS projects, *study* the project as a worked example, *add* minor enhancements, *build* larger components, and finally *leverage* FOSS for other purposes.

This paper describes experiences using FOSS and this approach in a sophomore-level SE course and in capstone projects. Section 2 briefly reviews SE course and project design issues, as well as FOSS and ways in which people relate to FOSS. Section 3 outlines the five step USABL model, and describes activities, assignments, and projects involving the model. Section 4 concludes with discussion of benefits and future directions.

2. Background

2.1. Pedagogical Challenges

Van Merrienboer and Kester[25] review and summarize an extensive literature on instruction design with a set of principles, such as:

- *Sequencing*: Sequence tasks from simple to complex.
- *Completion*: First study worked examples, then complete partial solutions, and finally, generate whole solutions.
- *Individuality*: Adjust task difficulty and support to the learner's ability and learning style.
- *Self-pacing*: Give learners control over the pace of instruction.
- *Variability*: Use a variety of tasks and contexts to help learners develop more general schema that can be transferred to other situations.

These principles can help instructors to design more effective projects and project-based courses. For example, instructors must decide whether students should design and implement an entire system, or extend and enhance one or more existing systems. In the former approach, students often see a wider variety of activities (encouraged by *variability*) but the scope is limited and the students may have to make critical decisions early in the project based on insufficient experience (violating *sequencing*). Too often, such projects are not really finished, but abandoned when the course ends. While it is valuable for students to build an entire system, it may be better to start with worked examples and partial solutions (suggested by *completion*), although students may see less of the early development activities and may have less choice of topic (violating *variability* and *individuality*). If students work exclusively on one project, they may have difficulty transferring what they have learned to other settings; on the other hand, each new project brings overhead that may distract or confuse students.

Instructors must choose the size of teams in which students will work. In smaller teams, students have more *variability* of roles and responsibilities, and *self-pacing* may be more feasible. In larger teams there is more opportunity for specialization and *individuality*, but it may be more difficult for students to see the big picture. Larger teams often present greater communication challenges, so *sequencing* from smaller to larger teams may be helpful. Often, however, courses emphasize communication within the team, rather than with users and other stakeholders outside of the formal development organization.

Instructors must choose the topics and activities to be emphasized, and the order in which they should appear. For a standalone project, students must necessarily start at the beginning with analysis and design, and may not make it to later activities involving development, testing, and maintenance, where the consequences of early decisions become more apparent. Instead, it may make more pedagogical sense to apply *sequencing* and *completion* and reverse the sequence, and have students start with maintenance and testing, and progress backwards to design and analysis[4,8,22]. This approach is more feasible when students work on existing projects.

2.2. Free and Open Source Software

Free and Open Source Software (FOSS) can help to address some of these challenges. FOSS refers to software that is distributed without charge and with the original source code, so that anyone can fix defects, add enhancements, or otherwise modify the software and share their changes with others. FOSS can benefit SE projects in several ways. First, successful FOSS usually has a robust implementation, including high cohesion, low coupling, and effective tests and documentation, since many developers work on them briefly or intermittently. Second, large FOSS projects often have varied user communities, which demonstrate the role and value of communication and supporting tools, such as discussion forums, version control, and task or defect tracking systems. Third, students may already be familiar with FOSS course management systems (e.g. Moodle, Sakai), content management systems (e.g. Drupal, Joomla), and project management tools (e.g. Bugzilla, Trac). Other researchers have examined uses of FOSS processes[5,11,19,23,14] and tools[10,18,24] in SE education, and other uses in FOSS in education [1,2].

FOSS communities are often represented as "onion" diagrams or pyramids[13]. Typically, the largest, outermost group is people who use or monitor the project, but do not contribute to it. Within this group are progressively more active but smaller groups, such as users who actively contribute ideas and defect reports, developers who work on specific sub-projects or supporting modules, leaders of sub-projects, and finally the core developers and overall project leaders. In general, new members of a particularly FOSS project start at the outside and work their way inward. My own experiences with FOSS generally parallel this sequence. There is a wide variety of FOSS that I use but have little or no other involvement with, and a small number of projects to which I have contributed defect reports or minor enhancements, such as scripts and macros. In a few cases I have made more significant contributions, such as plugins, supporting utilities, or device drivers. However, I have not (yet) become a core developer or project leader. Instead, I have done consulting and training for commercial and not-for-profit organizations. Increasingly, FOSS projects (such as wikis and content management systems) provide invaluable leverage, by allowing organizations to quickly and easily deploy general-purpose features (such as blogs, discussions, FAQs, and polls) and focus limited development resources on requirements that are unique to that organization.

Such progressions also apply to students, with an added benefit: FOSS provides worked examples to help them understand how software works and the implications of design decisions.

3. Approaches & Results

Thus, this paper proposes a 5 step model for integrating FOSS in CS and SE education. First, students *use* FOSS projects, without caring about the source code. This gives them a high-level understanding of what the project does, and may also lead them to identify problems or opportunities for improvement they can explore later. Second, students *study* the project as worked example. This can demonstrate the advantages of documentation, coding standards, modular design, and other practices. This can also help them understand the implications of various design and implementation choices. Third, students can *add* minor enhancements; many projects are designed to facilitate such changes. Fourth, students can *build* more complex components. Fifth, they can *leverage* the FOSS project for other purposes. The following subsections describe how this Use-Study-Add-Build-Leverage (USABL) model can be applied.

3.1. Computer Science 1 & 2

We have a traditional introductory CS sequence, currently using Java. Students use a variety of FOSS tools, including Linux, Moodle (a course management system), a graphics library in CS1[12,16], and Eclipse in CS2. Many students notice the similarities and differences with commercial software, and a few are intrigued by "free", but "open source" doesn't matter much, except when the instructor makes a minor change to a FOSS tool and points this out to students. In CS2, students usually have a series of assignments involving an extended example, such as a quiz system[16]. In the first assignments, students are given partial programs (e.g. stubs, driver programs, or unit tests to complete. In later assignments, they revisit and refactor or enhance their earlier assignments. The final system typically involves around 30 classes, including unit tests. For many students, this is their first experience with understanding and modifying code, and is a step toward later study of larger FOSS projects.

3.2. Software Engineering

This course is usually taken by sophomores (the prerequisite is CS2). At the end of the course, students should be able to: 1) analyze, design, implement, test, and review non-trivial software projects; 2) work effectively in teams; 3) develop and review proposals and reports, both written and oral; and 4) explain and apply the relationships between SE and related topics, including business. The course uses FOSS tools, including SVN, Trac (ticket and wiki collaboration), FreeMind (mind mapping), phpxref (PHP code cross-referencing), and OpenWorkbench (project scheduling). The course has two interleaved sequences of classroom topics and assignments. The first sequence focuses on technical issues, students work in teams on increasingly complex software projects. The first few assignments introduce students to HTML, CSS, forms, and PHP. These are followed by two analysis assignments and an implementation assignment, described below. The second sequence focuses on non-technical issues, including project management and topics in business and entrepreneurship. Students follow a Stage Gate model[6] customized for academic settings[17] in which they brainstorm and successively refine concepts, written proposals, and oral presentations for software-based businesses. Initially, each student explores several concepts. At each stage, there are fewer concepts, with more students working on each, so that by the end of the course 3-4 students are working on each concept.

3.2.1. Analysis 1

Students are given the following assignment:

1. *Choose an open source PHP project [from selection].*
2. *Get the complete project source code.(e.g. download it, load it via SVN, or use phpxref.)*
3. *Choose one page to study in depth; it should involve changing content, not static text.*
4. *Save the HTML page source from a browser to a text file.*
5. *Learn and document as much as you can about how the page is generated, by studying and comparing the source code (including HTML and CSS) and the resulting HTML. What files are used? What data comes from databases, files, or other sources?*
6. *Create a written overview, UML class and/or interaction diagrams, or other diagrams to show what is happening.*
7. *Meet with the instructor to assess your work.*

Most students can analyze at least portions of what is happening, and the individual meetings quickly reveal who did or didn't invest appropriate effort, and provide opportunities to address problems individually. However, some students have difficulty getting started, or deciding how and where to focus their analysis.

In the future, I will try a "treasure hunt" variation inspired by a debugging assignment[21]. Students will be given a URL and resulting HTML page source with a few highlighted tags, and instructed to describe the sequence of function calls that produce those specific tags. This approach gives the students a more specific objective, and should also be easier to evaluate.

3.2.2. Analysis 2

Students are given the following assignment:

1. *Create a ticket for the assignment, assigned to yourself.*
2. *Create a list of proposed features in the wiki, with difficulty and excitement ratings.*
3. *Choose one feature to investigate (each person choose a different one).*
4. *Study the code to understand how to implement the proposed feature, and create a work breakdown structure (WBS). You are not (yet) expected to implement the new feature.*

Students find it easier to start on this assignment, since it builds on the previous assignment. Project planning is more difficult, although they have some WBS experience from earlier coding assignments and from the concept proposals in the non-technical sequence.

3.2.3. Development

Some of the proposed features turn out to be quite simple, while others prove to be quite challenging; separating the analysis helps to identify features that present appropriate challenges based on student background, ability, and other activities in the course. Ideally, students would add one simple feature and then build something more complex; thus far, most students have done one or the other. Recent examples include:

- minor changes to the phpBB bulletin board system
- enhanced navigation, and due date and submission rate display in Moodle
- a configurable search engine block for Moodle

These experiences give students a better understanding of particular FOSS projects and software in general. Until this point, many students have a strong bias towards doing everything themselves, which can be an appropriate strategy in CS1 and CS2; this detailed exposure to the capabilities and inner workings of FOSS helps students begin to appreciate some of the challenges in larger development projects, and the leverage that can be provided by FOSS. For example, a recent team developed a concept for a web site to connect people and groups (such as clubs or Greek organizations) interested in volunteering with organizations that are seeking volunteers. The students quickly saw how Drupal (a content management system) would enable them to prototype most the site in a matter of days, allow them to focus on core functions unique to their project, and make it easy to add other collaboration and social networking features later.

In the future, I may add an assignment in which students develop unit tests for specific pieces of a FOSS project. This would give students more experience with language syntax, detailed understanding of the specified pieces, and possibly a better sense of overall organization. Thus, I would initially position this assignment between the analysis assignments described above.

3.3. Capstone & Projects

The capstone seminar includes advanced study of selected topics in computer science, and a significant software project, done alone or in teams. Advanced topics are explored through readings selected by students and the instructor. The first set of readings involve FOSS[3,9,13,20]. Projects can involve academic research (for students considering graduate school), extending or enhancing FOSS, or developing stand-alone software. Proposals and final projects are evaluated based on effort, what is produced, novelty, and relevance. This course also uses FOSS tools, particularly SVN and Trac. The course timeline is summarized below:

When	What
week 1	brainstorm ideas
week 2	submit concepts
week 4	submit proposal
weekly	complete 4+ project tickets per person
biweekly	project status report
last week	present project & submit final report

Some students still prefer to create something from scratch, but most choose projects that build on or leverage FOSS. This may be due to the increased emphasis on FOSS described above, or to the higher profile of FOSS in general. Recent examples include:

- A plugin for Pidgin (an multiprotocol instant messaging client) to create auditory alerts using eSpeak (a speech synthesizer). The plugin is available on SourceForge.
- Enhancements to SubjectsPlus (a library subject guide system), based on an earlier student project evaluating FOSS tools for academic libraries.
- Extensions so that Drupal (a content management system) can exchange product information and customer orders with OpenBravo (an ERP system).
- Improvements to the View project within Drupal to provide more powerful query capabilities, which will make it easier to develop a tutor management system.

4. Conclusions

The model described above is inspired by FOSS role progression and my own experiences. It seeks to utilize established instructional design principles to help students learn about broader ideas and issues in CS and SE. In accordance with the *sequencing* principle, students start with simple tasks such as using and tracing FOSS, and progress to more complex tasks such as building and leveraging. In accordance with the *completion* principle, students start by using and studying worked example, and progress to building or leveraging partial and complete solutions. Utilizing multiple FOSS projects adds *variability* and provides more opportunities for *individuality* and *self-pacing*.

Anecdotal data suggests that this approach is working well, although small class sizes (<10) make quantitative analysis difficult. However, at the end of both courses students write short reflective essays, which suggest that the approach described in this paper does support the goals of each course. Excerpts from Software Engineering essays:

> "Having to work with others on most projects was also something new to me … I found it hard to keep a balance between needing to know what was going on and just assuming things would all end up fine – I often felt unsure about whether or not I was doing enough for the group, or whether the project would be completed … I certainly learned that I need to keep a better focus on what everyone is doing."

> "All in all, I'm quite glad I took this course because it not only taught me about how to actually work as a programmer, but taught me useful organizational skills for completing any sort of project or assignment I may have."

Excerpts from capstone seminar essays:

> "I got some good exposure to open source communities and the ways that they work. … the contrast between [two different] communities allowed me to see just how important that momentum is for other developers."

> "This acted as a way to teach me how to work on a project for a customer. [We] were the contractors and the Librarians were our clients. We were consultants, providing progress reports and working on new acceptable features, trying to meet the clients needs. [We] were lucky to have a way to experience working with people who had a vested interested in our production."

As noted above, there are some specific changes and additions I plan to explore. I will try a "treasure hunt" assignment where students describe the sequence of function calls that produce specific HTML tags. I may add a coding assignment in which students develop unit tests. For the assignments that appear to be working well, I plan to refine the instructions, develop evaluation rubrics, and consider how to scale them to courses with larger enrollments. I would also like to find colleagues at other institutions interested in using and evaluating these approaches. Finally, with a larger sample of courses, students, and FOSS projects, it might be possible to analyze how student outcomes correlate with project characteristics such as the size of the FOSS project and the supporting community; the use of formal design documents, and the use of unit tests.

References

1. Amatriain, X., and Griffiths, D. 2004. Free Software in Education: Is it a Viable Alternative. In Proceedings of the 7th IMAC Conference. (Duisburg, Germany, September, Sept 13-15, 2004).
2. Attwell, G. 2005. What is the significance of open source software for the education and training community? In Proceedings of the First International Conference on Open Source Systems (Genova, Italy, July 11-15, 2005) 353-358.
3. Benkler, Y. 2005. Coase's penguin, or, Linux and the nature of the firm. In CODE: Collaborative Ownership and the Digital Economy, 169-206. Cambridge, MA: The MIT Press.
4. Buck, D., and Stucki, D. J. 2000. Design early considered harmful: Graduated exposure to complexity and structure based on levels of cognitive development. SIGCSE Bulletin 32, 1, 75-79.
5. Budd, T. 2009. A course in open source development. Integrating FOSS into the Undergraduate Computing Curriculum, Free and Open Source Software (FOSS) Symposium (Chattanooga, TN, Mar 4, 2009).
6. Cooper, R. G. 2001. Winning at New Products: Accelerating the Process from Idea to Launch. Perseus Books.
7. Fincher, S., Petre, M., and Clark, M. 2001. Computer Science Project Work: Principles and Pragmatics. Springer.
8. Gannod, B., Koehnemann, H., and Gary, K. 2006. The Software Enterprise: Facilitating the industry preparedness of software engineers. In Proceedings of the 2006 American Society for Engineering Education (ASEE) Annual Conference & Exposition (Chicago, IL, June 18-21, 2006).
9. Ghosh, R. A. 2005. Cooking-pot markets and balanced value flows. In CODE: Collaborative Ownership and the Digital Economy, 153-168. Cambridge, MA: The MIT Press.
10. Hartness, K. T. N. 2006. Eclipse and CVS for group projects. Journal of Computing Sciences in Colleges 21, 4 (Apr), 217-222.
11. Horstmann, C. 2009. Challenges and opportunities in an open source software development course. Integrating FOSS into the Undergraduate Computing Curriculum, Free and Open Source Software (FOSS) Symposium (Chattanooga, TN, Mar 4, 2009).
12. Huggins, J. et al. 2003. Multi-phase homework assignments in CS I and CS II. Journal of Computing Sciences in Colleges, 19 (Dec), 182-184.
13. Jensen, C., and Scacchi, W. 2007. Role migration and advancement processes in OSSD projects: A comparative case study. In Proceedings of the 29th International Conference on Software Engineering (Minneapolis, MN, May 19-27, 2007), 364-374. IEEE Computer Society.
14. Kamthan, P. 2007. On the prospects and concerns of integrating open source software environment in software engineering education. Journal of Information Technology Education 6, 45-64.
15. Kussmaul, C. 2008. Open source software to support student teams: Challenges, lessons, and opportunities. In Proceedings of the 2008 American Society for Engineering Education (ASEE) Annual Conference & Exposition (Pittsburgh, PA, June 22-25, 2008).
16. Kussmaul, C. 2008. Scaffolding for multiple assignment projects in CS1 and CS2. In Proceedings of the ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (Nashville, TN, Oct 19-23, 2008).
17. Lane, P, Farris, J., and Kussmaul, C. 2006. Where to begin and what to do next? Stages and gates for an interdisciplinary student population. In Proceedings of the NCIIA 10th Annual Meeting (Portland, OR, Mar 23-25, 2006) 189-198.
18. Liu, C. 2005. Using issue tracking tools to facilitate student learning of communication skills in software engineering courses. In Proceedings of the 18th Conference on Software Engineering Education & Training (Ottawa, Canada, April 18-20, 2005), 61-68.
19. Morelli, R., and de Lanerolle, T. 2009. FOSS 101: Engaging introductory students in the open source movement. In Proceedings of the 40th SIGCSE Technical Symposium on Computer Science Education (Chattanooga, TN, Mar 3-7, 2009), 311-315.
20. O'Mahony, S., and Ferraro, F. 2007. The emergence of governance in an open source community. The Academy of Management Journal 50, 5, 1079-1106.

21. Richards, B. 2000. Bugs as features: Teaching network protocols through debugging. In Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education (Austin, TX, Mar 8-12 2000), 256-259. ACM.
22. Sebern, M. 2002. The software development laboratory: Incorporating industrial practice in an academic environment. In Proceedings of the 15th Conference on Software Engineering Education and Training, 2002 (Covington, KY, Feb 25-27, 2002), 118-127.
23. Seiter, L. 2009. Computer science and service learning: Empowering nonprofit organizations through open source content management systems. Integrating FOSS into the Undergraduate Computing Curriculum, Free and Open Source Software (FOSS) Symposium (Chattanooga, TN, Mar 4, 2009).
24. Toth, K. 2006. Experiences with open source software engineering tools. IEEE Software 23, 6, 44-52.
25. Van Merrienboer, J. J. G., and Kester, L. 2005. The Four-Component Instructional Design Model: Multimedia principles in environments for complex learning. In The Cambridge Handbook of Multimedia Learning, ed. Richard E. Mayer. Cambridge University Press.