Free and Open Source Software across the Curriculum: Use, Study, Add, Build, Leverage

Clifton Kussmaul
Muhlenberg College
2400 Chew St
Allentown, PA 18018
1-484-664-3352

kussmaul@muhlenberg.edu

ABSTRACT

This position paper argues that FOSS should be used in a variety of different ways across the undergraduate CS curriculum, and particularly in projects. The paper describes a five step **USABL** model in which students **use** FOSS, **study** it as a worked example, **add** minor enhancements, **build** larger components, and finally **leverage** FOSS for other purposes. This position is based on experiences using FOSS in a variety of contexts, particularly a software engineering course and in capstone projects.

Categories and Subject Descriptors
K.3.2 [**Computer and Information Science Education**]: Computer science education.

General Terms
Management, Documentation, Design, Economics

Keywords
Free and Open Source Software, Stage Gate

1. INTRODUCTION

Free and Open Source Software (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.

This position paper argues that FOSS should be used in a variety of different ways across the undergraduate curriculum, and particularly in software projects. This position is based on experiences in multiple contexts, particularly a software engineering course and in capstone projects.

2. A "USABL" MODEL

We propose a 5 step model for integrating FOSS in CS and SE education. The model described above is inspired by role progression within FOSS (e.g. [5]), effective practices for instructional design (e.g. [10]), and by the author's experiences. 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; as noted above, many projects have architectures that facilitate such changes. Fourth, students can **build** more complex components. Fifth, students can **leverage** the FOSS project for other purposes. The application of this **Use-Study-Add-Build-Leverage** (USABL) at Muhlenberg College is described below.

2.1.1 CS1 and CS2

In CS1, students use a variety of FOSS tools, including Linux, the Moodle course management system, and a graphics library [4][6]. Many students notice the similarities and differences with commercial software, and a few are intrigued by "free", but "open source" doesn't matter much.

In CS2, students use Eclipse, and have a series of assignments involving an extended example, usually a quiz system [6]. At first, 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 work. The final system typically involves around 30 classes, including unit tests. This is often students' first experience with understanding and modifying code, and a key step toward studying larger FOSS projects.

2.1.2 Software Engineering

In Software Engineering, students use a wider variety of FOSS tools, including SVN, Trac (ticket and wiki collaboration), FreeMind (mind mapping), phpxref (code cross-referencing), and OpenWorkbench (scheduling). The course has two interleaved sequences of classroom topics and assignments. One sequence focuses on non-technical issues, including project management and topics in business and entrepreneurship. Students use a customized Stage Gate model [2][7] to successively refine concepts, written proposals, and oral presentations for software-based businesses. The other sequence focuses on technical issues, and 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.

In the first analysis assignment, each student **studies** a web page dynamically generated by FOSS, to learn and document how part of the system works. Each student creates a written overview with supporting diagrams, and meets with the instructor to assess their work and address problems individually. In the second analysis assignment, each student **studies** how to implement a specific feature, and creates a corresponding work breakdown structure (WBS). In the implementation assignment, students add some of these features, often in small teams. Some of the features turn out to be quite simple, while others are quite challenging; separating the analysis helps to identify appropriate features. Ideally, students would first **add** simple features and then **build** more complex features; thus far, most students do one or the other.

Together, these analysis and implementation assignments give students a better understanding of particular FOSS projects and software in general. Students find it easier to start these assignments, since each one builds on previous experiences. This exposure also helps students begin to appreciate the **leverage** that FOSS can provide. For example, students quickly saw how the Drupal content management system enabled them to prototype a site for a business concept in a matter of days, help them to focus on core functions, and allow them to add other features later. In the future, I might separate the analysis assignments with a assignment in which students develop unit tests for 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.

2.1.3 Capstone & Projects

The capstone CS seminar includes advanced study of selected topics, 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 (e.g. [1][3][5][9]). Projects are evaluated based on effort, what is produced, novelty, and relevance. Projects can involve academic research (for students considering graduate school), developing stand-alone software, or extending or enhancing FOSS; most students choose the latter. Recent projects include:

- Enhancements to the SubjectsPlus library subject guide system, based on an earlier student project evaluating FOSS tools for academic libraries [8].
- A plugin for the Pidgin multiprotocol instant messaging client to create auditory alerts using the eSpeak speech synthesizer. The plugin is available on SourceForge.
- Extensions so that Drupal can exchange product information and customer orders with the OpenBravo ERP system.

3. CONCLUSIONS

The symposium CFP identified three key questions.

Can FOSS increase student participation in CS by involving them in engaging and intellectually challenging real-world software projects? In my experience, FOSS can help involve students in such projects, although it is not clear whether this is an effective recruiting tool.

Is student interest in Wikipedia, YouTube, and other aspects of today's "openness" an effective vehicle to connect the computing major with larger numbers of undergraduates? Unfortunately, many people do not appreciate the "openness" of such systems, and view them – like word processors, spreadsheets, and cars – as useful tools, but not as inherently interesting systems.

Can the cooperative and community aspects of FOSS projects help to attract and retain students from traditionally underrepresented groups? Yes, to the extent that such groups value cooperative and community. On the other hand, students traditionally attracted to CS may be turned off by the same aspects.

However, it is clear that FOSS is valuable in CS and SE education for other reasons. The USABL model described above seeks to utilize established instructional design principles to help students learn about broader ideas and issues in CS and SE. Anecdotal data suggests that this approach is working well, although small class sizes (<10) makes it hard to assess this quantitatively. I would like to find colleagues at other institutions interested in using and evaluating these approaches.

4. REFERENCES

[1] 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.

[2] Cooper, R. G. 2001. Winning at New Products: Accelerating the Process from Idea to Launch. Perseus Books.

[3] 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.

[4] 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.

[5] 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.

[6] 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).

[7] 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.

[8] Lobdell, C., and Kussmaul, C. 2009. Free/open source software tools for a library website: A comparative analysis. EDUCAUSE Mid-Atlantic Regional Conference (Philadelphia, PA, January 7-9, 2009).

[9] 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.

[10] 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.