An Approach for Evaluating FOSS Projects for Student Participation

Heidi J. C. Ellis

Western New England University
1215 Wilbraham Rd.

Springfield, MA 01119

00-1-413-782-1748

ellis@wne.edu Michelle Purcell

Drexel University
3141 Chestnut St

Philadelphia, PA 19104

00-1-215-895-2179

mjw23@drexel.edu

Gregory W. Hislop

Drexel University
3141 Chestnut St

Philadelphia, PA 19104

00-1-215-895-2179

hislop@drexel.edu

**ABSTRACT**

Free and Open Source Software (FOSS) offers a transparent development environment and community in which to involve students. Students can learn much about software development and professionalism by contributing to an on-going project. However, the number of FOSS projects is very large and there is a wide range of size, complexity, domains, and communities, making selection of an ideal project for students difficult. This paper addresses the need for guidance when selecting a FOSS project for student involvement by presenting an approach for FOSS project selection based on clearly identified criteria. The approach is based on several years of experience involving students in FOSS projects.

**Categories and Subject Descriptors**

K.3.2 [**Computers and Education**]: Computer and Information Science Education – *Computer science education.*

**General Terms**

Management, Human Factors.

**Keywords**

Free and open source software, computing education, student projects

# PROBLEM STATEMENT

The use of free and open source software (FOSS) in academia is growing as instructors realize the rich learning environment offered by the transparency of the development process and artifacts. FOSS projects may be used to support a wide variety of learning objectives. Such projects are frequently used as a tool to teach computing topics such cryptography [1], distributed software development [2], operating systems [3], and robotics [4]. Frequently, FOSS code is used as a base for learning [5-6].

This paper focuses on the direct participation of students in FOSS projects that involves making some form of contribution to the project. Several recent efforts have involved students in FOSS projects within the classroom [7-14]. While these reports of student learning in FOSS projects are positive, many instructors are cautious in adopting this approach. FOSS projects can be complex and may present learning challenges to both instructors and students including:

* presence of a large, complex code base
* use of different or new programming languages
* use of a variety of development tools
* understanding of community interactions
* scheduling of deliverables with FOSS calendar

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*Conference’10*, Month 1–2, 2010, City, State, Country.

Copyright 2010 ACM 1-58113-000-0/00/0010…$10.00.

While some recent work has provided guidance to instructors desiring to involve students in FOSS projects [15, 16], approaches to identifying appropriate FOSS projects and selecting reasonable assignments is an ongoing issue. This paper attempts to address this problem by providing a framework of evaluation criteria that can be used to determine approachable FOSS projects that may support student involvement.

## Open Source Culture and Tools

In order for instructors to provide entre for students into FOSS projects, instructors must understand the FOSS culture and tools used. There are several excellent books that cover the open source culture [17-19]. A discussion of the steps that may be taken to involve students in FOSS projects from the perspectives of the FOSS and academic communities is included in [15].

There are three major resources that instructors may find helpful when involving students in FOSS projects. The first is the TeachingOpenSource [20] community which was established to aid instructors who desire to involve students in FOSS projects. The second are forges. A software forge such as sourceforge.net is a platform that supports collaborative software development. In addition, a forge will typically host many independent projects. Lastly, ohloh.net is a site that collects, collates and analyzes data from multiple FOSS projects. Ohloh provides information about the landscape of FOSS projects, allowing one to search and compare FOSS projects by project type, language and other characteristics.

One of the most important aspects for instructors to remember when involving students in FOSS projects is the principle of “giving back”. FOSS instructors must identify ways that students can make contributions to projects that are useful to the FOSS community. This is essential to supporting the community and provides an important life lesson to students.

# KEY EVALUATION ASPECTS

From the instructor’s perspective, there are several obvious characteristics of FOSS projects that can be used to evaluate the “fit” of a FOSS project for use with a class. These aspects include the suitability of FOSS project opportunities to support student assignments, platform used, programming language(s) used, development environment, and perhaps hardware requirements. These aspects tend to be well-understood by instructors and the fit of the project easily assessed based on these characteristics.

However, since the student will be involved in an ongoing FOSS project, the health, personality, and approachability of the project itself must be understood. Several recent research efforts have resulted in models for assessing the health of a FOSS project.

An early work by Crowston et al. [21] describes a set of measures of success for FOSS projects that include user satisfaction and involvement, product quality, software process, age of the project, number of developers and their satisfaction, and the number of downloads. Hahn et al. [22] identify the ability to attract volunteers and to organize contributions as a success factor for FOSS projects. Weiss [23] describes a model for measuring the success of FOSS projects based on the popularity of the project on the Web. The approach mines search engine knowledge to determine the popularity of a project. Beaver et al. [24] describe a model based on existing FOSS project data that uses agents to simulate behavior of developers. The model is intended to predict the success of the FOSS project based on individual contributions, actions of developers, and popularity and performance of the project. English and Schweik [25] describe a classification of FOSS projects based on characteristics of how the project was founded and its’ growth pattern afterwards.

While these efforts provide insight into the workings of FOSS projects, their complexity typically prohibits their use in identifying FOSS projects for possible student participation. Callaway [26] has created a metric based on product and development features that assigns points to various FOSS project features including bundling, libraries, source control, releases, documentation, licensing and more. The drawback to this method is that it is time-intensive. However, this might be a good exercise for students to do once a project has been selected. Below we suggest a straightforward approach for instructors to use to select FOSS projects appropriate for student involvement.

# A MODEL FOR EVALUATING FOSS PROJECTS

The first step in selecting a FOSS project is to identify a set of potential projects. One way to do this is to browse a forge such as sourceforge.net, launchpad.net, or gitorious.org. Ohloh provides a project profile for a FOSS project including programming language used, commit history, history of the codebase and more.

Begin by identifying a set of five or six potential projects. Once this set has been identified, select three of the most interesting. You may return to this set and choose another project if one in the subset does not meet the criteria at any phase.

An overview of the evaluation model is shown in Figure 1. The model evolved out of several years of experience involving students in FOSS projects and includes evaluation criteria for three important aspects of FOSS projects. A FOSS project’s **viability** is essential to supporting student learning. The project must have an active community that is engaged in ongoing development. Student participation is dependent on the **approachability** of the project. The project should have an identifiable on-ramp for new participants and describe how new contributors may join the community and make contributions. A third important aspect is that of **suitability.** The project must provide appropriate learning experiences for the course as well as a sufficient number of learning opportunities.

**Mission**

**Critical**

**Criteria**

**Viability**

**Suitability**

- Size/Scale/
 Complexity

- Activity

- Active
 Community

- Clear
 On-Ramp

- Appropriate
 Artifacts

- Contributor
 Support

- Domain

- Maturity

- User Support

- Roadmap

- Contribution
 Types

- Openness to
 Contributions

- Student
 Friendliness

- Product
 Description

- Platform

- Development
 Features

**2nd-ary Criteria**

**Approach-
ability**

Figure 1. FOSS Evaluation Criteria.

The model differentiates between two levels of criteria. **Mission Critical** criteria are characteristics of the FOSS project that must be present in order to support student success. **Secondary Criteria** are criteria that contribute to the success of student experiences, however a FOSS project may fail to meet one or two of these criteria and still be a viable project for student participation.

The model is intended to be applied in a top-down fashion where the **critical** aspects are evaluated first. This allows instructors to more easily weed out projects that are infeasible for student involvement.

In the following sections, we describe each of the three evaluation aspects and their criteria. We tag each criterion as being either mission critical or secondary and provide general guidelines for how to determine whether a criterion has been met.

## Viability

This set of criteria is intended to help identify the maturity, stability and future prospects of a FOSS project. These criteria help identify projects that are still in very initial stages of development, very complex projects, dormant projects, and projects that lack sufficient development community. Much of the data used in this evaluation may be obtained from the project web site, Ohloh.org., or SourceForge.net.

**Size/Scale/Complexity (critical)** – An ideal project should be neither overly simple nor overly complex. Students do not need to understand the entire project, however they should be able to understand a subset of the project. One quick measure of size is LOC. For instance, smaller FOSS projects include the Maven Eclipse Plugin at 96 KLOC and the Eclipse Checkstyle Plugin at 60 KLOC. A medium size FOSS project that has been used successfully in classes by the authors is OpenMRS which has 213 KLOC. Larger projects include Chromium at 5 MLOC and Mozilla at 5.3 MLOC.

However, LOC does not provide insight into the architecture of the project, nor its modularity. Larger projects such as Mozilla may use a plug-in architecture that supports small student code contributions. Therefore, observation of the design provides insight into the modularity and complexity of the project.

Another indication of project complexity is the number of committers on the project. A reasonable number is at least six committers working on the project within the last 12 months. If a project is much smaller than this, the project may not be sufficiently mature to provide a rich learning environment. If the project is larger, the complexity of the code (and development environment) may present a barrier to student learning.

**Activity (critical) -** The project should be reasonably active. Ten commits per month is minimally acceptable and 30 commits or more per month, on average, would be favorable. Looking back a year or more will provide an understanding of the pattern of commits. This data can be retrieved from Ohloh.net.

**Community (critical)** – In order to support student learning, the project must have an active user community. While it is difficult to quantitatively evaluate the activity of a user community, some indicators of community health include a regular history of project downloads and documentation updates over time, current activity on user mailing lists, and testimonials on the project web site.

It is also important to observe the recency of the activity. Are there lags between updates? Are there current questions that have not been answered on forums or lists? What is the IRC activity? What is the recent history of downloads? All of these will provide insight into the community activity.

**Domain (secondary) -** The domain of the application may have an impact on student learning. Some FOSS projects require a lot of domain knowledge. For instance, gaming software likely presents less of a learning curve to students than software to process data from nuclear magnetic resonance experiments.

**Maturity (secondary) –** The project should have at least one stable production release. Projects that lack this may not have the organization to support student learning. SourceForge.net provides a *status* feature that indicates releases and Ohloh.net provides an *analysis summary* that provides information on the stability of the code base.

**User support (secondary) –** The project should have clear instructions for downloading, installing and using the project. Other evidence of user support includes FAQs, forums, and lists for users. Such support for users of the project gives some indication of responsiveness of developers to users and to activity in the project. The quality of end-user documentation also provides insight into the maturity of the project.

**Roadmap for the future (secondary) –** Does the project have a roadmap or blueprint for future development? Student learning is best supported by projects that have a roadmap that includes new feature development, a method for users to submit new feature requests and a process for identifying how new features are prioritized.

## Approachability

In addition to a project being viable, in order to successfully support student participation, the project must be approachable by students. It must have sufficient documentation and guidance for students, and the community must also be welcoming for new contributors. Specific criteria for determining project approachability include:

**On-ramp (critical)** - The project must have an identifiable "on-ramp" for new contributors to become involved and to learn. The project should contain instructions for how to join the community and first steps toward making a contribution. Some projects may even have on-ramp instructions aimed specifically for students. We suggest the following 3-point scale for evaluating on-ramp:

* Insufficient-Few or no pointers on how to get involved.
* Sufficient--Suggestions about how to get involved other than contributing money with accompanying high-level instructions.
* Ideal -Obvious link to get started, list of suggestions for things to do and detailed instructions.

**Contribution types (secondary) -** Does the project contain opportunity for multiple types of contributions (documentation, requirements, testing, debugging, coding, etc)? Clearly, the project should include opportunities for the type of contribution that fits the class.

**Openness to contributions** **(secondary)** – Acceptance of a student contribution to a project provides valuable affirmation to student learning. Determine whether the project accepts external patches. Ask how to get student work committed or look to see if the community has a description of how contributions are accepted or whether a “committer” is identified. A project that never commits any of the code from outside the core community members is likely to not accept new contributions from students which may negatively affect student motivation.

**Student-friendliness (secondary) –** An ideal project for student involvement is one that welcomes and values student contributions. One easy gauge of the student-friendliness of a project is if the community members moderate the tone of communication. Review developer mailing lists and archives and sit in on an IRC conversation to gauge tone. If developers are getting “flamed” that will have a negative effect on student motivation.

## Suitability

The suitability of a FOSS project depends a large part on the course and its content. Instructors should reflect on the learning objectives for the course and the types of activities that could be undertaken as well as the types of contributions that students could make. For instance, a course on technical writing would be a good fit for a FOSS community that needs developer or user documentation written. Similarly, a web-based FOSS project might serve as a good project for a usability course.

**Appropriate artifacts (critical) –** FOSS projects offer a wealth of different types of contributions that may serve as a base for student work. For instance, many FOSS projects need documentation written, testing performed, or designs created, in addition to code contributions. In order for a particular FOSS project to be suitable for a class, appropriate artifacts for students to both study and contribute to must exist in that FOSS project. Therefore, a FOSS project being evaluated for use in a course on technical writing should provide multiple opportunities for students to contribute documentation.

**Contributor support** **(critical)** – The ideal FOSS project for student participation is one that contains a high volume of guidance for students as they learn. The project should contain information on how the project is administered and managed. Communication tools such as IRC and lists should be clearly documented and developers should have a web presence. The operating processes such as the process for getting a change committed, feature selection and approval, and feature status should be well documented. In addition, the questions posed on IRC, lists, and forums should have supportive and timely responses.

**Product description (secondary) –** In order for a project to support student participation, students must be able to understand the purpose of the project. Does the project clearly describe the product? Can students understand the intended uses of the product?

**Platform** **(secondary)** – What software and hardware platform does the FOSS project run on? Are resources available to support these platforms? Typically instructors desire a platform with which students are familiar.

**Development features (secondary) –** Some classes may be dependent on features of the actual FOSS project code including:

* **Programming language** – Students may find it easier to participate in a project that uses a programming language familiar to them. FOSS projects may use more than one programming language so when you get to actual assignments, review that the programming language is a good fit.
* **Development environment** – Similar to programming language, students may find it easier to participate in a project that uses tools and a development environment with which they are familiar. This criterion may be less important than the programming language but for large projects, the development environment can be quite complex.
* **Supporting technologies** – Does the project depend on additional software packages such as databases or libraries? The presence of additional technologies adds to the complexity of a project and may create a steeper learning curve for students.

## Other Considerations

In addition to the criteria described in the previous three sections, there are several characteristics of projects that should be considered:

**Sizzle-factor –** How attractive is the project to students? Projects that students find more appealing (e.g., games) may provide greater motivation to students. Projects with a humanitarian focus may provide motivation for students [8, 10, 11].

**Long-term prospects** – Most instructors teach a similar set of courses from term to term and from year to year. Selection of a project that has the possibility of supporting student participation for future terms and years may reduce instructor preparation time and effort for future courses.

**Ownership -** Does the project hold the possibility for being able to create a sense of “ownership” for students? Is this a project that students will want to follow during future development? Creating a sense of project ownership in students may also increase student motivation and learning.

# APPLICATION OF THE APPROACH

In this section, we show how the evaluation approach described in the previous section is applied to example projects. Given the variety of courses that may desire to involve students in FOSS projects and the possible variety in topics, we do not provide concrete rubrics for evaluating each criterion. Instead, we suggest that each criterion be rated on a one-to-three scale where one indicates that the criterion is lacking in the FOSS project and three indicates full support for the criterion by the FOSS project. The general approach is:

1. Evaluate the **mission critical** criteria first. A scale of less than a two on any of these criteria should remove the FOSS project from consideration.
2. Evaluate the **secondary** criteria on a scale from one to three. A score of above 20 for the **secondary** criteria indicates a viable project. Projects that score below 20 should be examined to determine if the low-scoring criteria will negatively impact student involvement.

The first evaluation is of the Mifos microfinance application, which provides a framework of functionality for institutions including client management; portfolio management; and loan tracking, supporting 28 microfinance institutions. Three sites were used during the assessment including the Mifos site (<http://mifos.org/>), the Mifos SourceForge site (<http://sourceforge.net/projects/mifos/>), and the Ohloh page on Mifos (<http://www.ohloh.net/p/mifos>).



Figure 2. Evaluation of Mission Critical Criteria for Mifos.

Figure 2 above shows the result of step 1, evaluation of the mission critical criteria. The figure shows both the numeric evaluation as well as comments that provide some rationale for the rating. Based on the results of the evaluation, the Mifos application fulfills all of the mission critical criteria. The strong positive aspects of Mifos include a clear on-ramp for new contributors including instructions and the presence of useful documentation on coding standards and how to submit patches.

As none of the mission critical criteria rank less than a two (see Figure 2), the secondary criteria are then evaluated. Figure 3 shows the result of the evaluation.



Figure 3. Evaluation of Secondary Criteria for Mifos.

The evaluation results shown in Figure 3 appear to indicate that Mifos is a very good project for student involvement. Mifos is a mature project with stable releases and good user and developer support. There are multiple ways that students can contribute to the project and the community is open to student participation. The domain is understandable by students and the development platform requires no special resources.

In order to demonstrate evaluation of a project that may have potential, yet is not yet ready for student involvement, Figure 4 shows the evaluation of the Mission Critical criteria for Sigmah. Sigmah is a Humanitarian Project Management application (<http://www.sigmah.org/>). Upon first look, Simgah is appealing based on its size and complexity and level of activity. However, the number of product downloads are few and the user guide is only available in French which could indicate a small user base. This also makes it difficult for students to understand the project, especially since most of the demonstration site is also in French.



Figure 4. Evaluation of Mission Critical Criteria for Sigmah.

It should be noted, however, that this project may evolve to be one that could support student involvement. The project is relatively young, the domain is reasonably understandable by students, and the development team has already provided some good resources to help developers get started. In addition, a web search reveals that Sigmah has 12 non-governmental organizations in France on their steering committee which could indicate a stable future for the project. This project may pick up momentum and be one where student involvement can make a real difference.

# CONCLUSION

This paper presents an approach for evaluating FOSS projects for student involvement and contribution to the project. Criteria that impact student involvement in a project are identified and a set of Mission Critical criteria that must be fulfilled are presented. The Mifos microfinance and the Sigmah humanitarian project management software are used as examples to demonstrate the utility of the approach. The model and approach described in the paper evolved out of several years of experience involving students in FOSS projects and is flexible to support differing types of student participation including fixing bugs, writing code, writing documentation, testing, and more.

# ACKNOWLEDGMENTS

This material is based on work supported by the National Science Foundation under Grants DUE-0958204, DUE-0940893, and DUE-0940925. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF). We would also like to thank Drexel students Andrew Hagner, Benjamin Kos, and Jon Thompson for applying the model to FOSS projects.

# REFERENCES

1. McAndrew, A. 2008. “Teaching cryptography with open-source software, *Proceedings of the 39th SIGCSE technical symposium on Computer science education, SIGCSE '08,* pp. 325—329, doi =http://doi.acm.org/10.1145/1352135.1352247.
2. Meneely, A. and Williams, L., 2009. “On preparing students for distributed software development with a synchronous, collaborative development platform,” *SIGCSE Bull.*, Vol. 41, No. 1. Mar. 2009, pp. 529--533, doi = http://doi.acm.org/10.1145/1539024.1509047
3. Nieh, J. and Vaill, C. 2005. “Experiences teaching operating systems using virtual platforms and linux,” *SIGCSE Bull.*, Vol. 37, No. 1, Feb. 20005, pp. 520--524, doi = http://doi.acm.org/10.1145/1047124.1047508
4. Nickens, G. V., Tira-Thompson, E. J., Humphries, T. and Touretzky, D. S., 2009. “An inexpensive hand-eye system for undergraduate robotics instruction,” *SIGCSE Bull.*, Vol. 41, No 1, Mar. 2009, pp.423--427, doi = http://doi.acm.org/10.1145/1539024.1509016
5. Petrenko, M., Poshyvanyk, D., Rajlich,V., and Buchta, J., 2007. “Teaching Software Evolution in Open Source,” *IEEE Computer*, Vol. 40, No. 11, pp. 25-31
6. Liu, P. L., 2008. “Using open-source robocode as a Java programming assignment,” *SIGCSE Bull.*, Vol. 40, No 4., Nov. 2008, pp. 63--67, doi =http://doi.acm.org/10.1145/1473195.1473222
7. Pedroni, M., Bay, T., Oriol, M., Pedroni, A., 2007. “Open Source Projects in Programming Courses”, *ACM SIGCSE Bulletin*, Vol 39, No 1., pp. 454-458.
8. Ellis, H.J.C., Morelli, R.A., de Lanerolle, T., 2007. “Holistic Software Engineering Education Based on an Open Source Project,” *20th Annual Conference on Software Engineering Education and Training*, pp. 327-335.
9. Hepting, D. H., Peng, L., Maciag, T. J., Gerhard, D., and Maguire, B. 2008. “Creating synergy between usability courses and open source software projects,” *SIGCSE Bull.*, Vol. 40, No. 2, June 2008, pp. 120-123, doi = http://doi.acm.org/10.1145/1383602.1383649
10. Hislop, G.W., Ellis, H.J.C., and Morelli, R.A., 2009. “Evaluating Student Experiences in Developing Software for Humanity”, *ACM SIGCSE Bulletin - ITiCSE*, Vol 41, No 3., pp. 263-267.
11. Morelli, R. A., Tucker, A., Danner, N., De Lanerolle, T R., Ellis, H. J. C., et al, 2009. “Revitalizing computing education through free and open source software for humanity”, *CACM,* Vol. 52, No 8., pp. 67-75.
12. Morelli, R. A., and De Lanerolle, T. R., “FOSS 101: Engaging introductory students in the open source movement,” 2009. *ACM SIGCSE Bulletin - SIGCSE '09*, Vol. 41, No 1., pp. 311-315.
13. Marmorstein, R., 2011. “Open source contribution as an effective software engineering class project,” Proceedings of the 16th annual joint conference on Innovation and technology in computer science education, ITiCSE '11 Darmstadt, Germany, pp. 268-272. doi = <http://doi.acm.org/10.1145/1999747.1999823>
14. Yue, K.B., Damania, Z., Nilekani, R. and Abeysekera, K., 2011. “The use of free and open source software in real-world capstone projects,” J. Comput. Sci. Coll., Vol. 26, No. 4, pp. 85-92.
15. Ellis, H.J.C., Hislop, G.W., Chua, M., and Dziallas, S., 2011.“How to Involve Students in FOSS Projects,” *41st Annual Frontiers in Education Conference, 2011. FIE 2011,* October 2011.
16. Professors Open Source Summer Experience, <http://www.communityleadershipteam.org/posse/> retrieved 8/10/11.
17. Raymond, E., 1999. “The Cathedral and the Bazaar”, *Knowledge, Technology & Policy,*Vol 12, No 3, pp. 23-49.
18. Feller, J., Fitzgerald, B., Understanding Open Source Software Development, Pearson Education, 2001.
19. Fogel, K., 2005. Producing Op Producing Open Source Software, How to Run a Successful Free Software Project, Retrieved 8/20/11 from: <http://producingoss.com/en/index.html>
20. teachingopensource.com
21. Crowston, K., Annabi, H., and Howison, J., 2003. “Defining Open Source Project Success,” In *Proceedings of the 24th Int.l* *Conference on Information Systems,* ICIS, Seattle.
22. Hahn, J., Moon, J.Y., and Zhang, C., “Emergence of New Project Teams from Open Source Software Developer Networks: Impact of Prior Collaboration Ties,”2008.  *Information Systems Research,* Vol. 19, No. 3, pp. 369-391. DOI: 10.1287/isre.1080.0192
23. Weiss, D., “Measuring Success of Open Source Projects Using Web Search Engines,” 2005. *The First International Conference on Open Source Systems OSS2005: Open Source Systems*, Geneva, Italy, pp. 93-99.
24. Beaver, J.M., Cui, X. St Charles, J. L. and Potok, T. E., 2009. “Modeling success in FLOSS project groups,” *Proceedings of the 5th International Conference on Predictor Models in Software Engineering, PROMISE '09,* Vancouver, British Columbia, Canada, pp. 16:1--16:8. doi = <http://doi.acm.org/10.1145/1540438.1540461>
25. English, R., Schweik, C. M., “Identifying Success and Tragedy of FLOSS Commons: A Preliminary Classification of Sourceforge.net Projects, 2007. ” *First International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS'07: ICSE Workshops 2007)* Minneapolis, Minnesota, May 20-May 26.
26. Callaway, T., “How to Tell if a FLOSS Project is Doomed to Fail,” <http://www.theopensourceway.org/wiki/How_to_tell_if_a_FLOSS_project_is_doomed_to_FAIL> retrieved 8/10/11.