Free/Libre/Open Source Software Development in Software Engineering Education: Opportunities and Experiences

Christina Chavez[1], Antonio Terceiro[1], Paulo Meirelles[2], Carlos Santos Jr.[2], Fabio Kon[2]

[1]Software Engineering Labs
Department of Computer Science - IM
Federal University of Bahia (UFBA), Brazil

{flach,terceiro}@dcc.ufba.br

[2]FLOSS Competence Center
Department of Computer Science - IME
University of São Paulo (USP), Brazil

{paulormm,denner,fabio.kon}@ime.usp.br

Abstract. *Free/Libre/Open Source Software (FLOSS) presents a strategy for developing software products that is substantially different from what is usually taught in Software Engineering courses. This paper discusses the benefits of using FLOSS in Software Engineering Education, proposes a list of topics that should be covered in FLOSS-based Software Engineering courses and briefly reports our experience at two Brazilian universities.*

1. Introduction

Free/Libre/Open Source Software (FLOSS)[1] represents a change in the way software is designed, constructed, and evolved. In the last decades we have witnessed FLOSS projects become increasingly important in society: Apache HTTP Server omnipresence in the web server environment; the increasing market share obtained by Firefox, despite the fact that its main competitor comes pre-installed with the large majority of personal computers sold; Eclipse large penetration in Software Engineering (SE) research.

Most FLOSS projects are developed in the open: the source code is public, there are mailing lists for discussions between developers, the bug tracking systems are available and usually there is a documentation website where beginners can learn about how to get started with development in the project. This environment opens up a lot of opportunities for Software Engineering Education (SEE). Thus, top international forums such as the Software Engineering Education track at ICSE (SEE&T), the Academy for Software Engineering Education and Training (ASEE&T), and the Frontiers in Education Conference (FIE) have been discussing how to take advantage of such opportunities.

To investigate ongoing initiatives from the Brazilian SEE community on the use of FLOSS, we performed an study on the papers from FEES (Fórum de Educação em Engenharia de Software), the Brazilian forum co-located with SBES (Brazilian Symposium on Software Engineering), that brings together researchers interested in SEE since 2008. We analyzed the set of 30 accepted papers by the three FEES editions and observed

[1]In this work, the acronym "FLOSS" is used as a representative for "Free Software", "Open Source Software" (OSS) and "Free/Open Source Software" (FOSS).

that no paper from this set addressed the use of FLOSS in SEE. To close this gap with respect to the FEES forum, in this paper we (i) explore some opportunities brought by the usage of FLOSS in SEE, (ii) briefly discuss our previous experience with FLOSS-based SE courses, and (iii) propose a list of sample topics that can be included in FLOSS-based SE courses. This way, we expect to promote an initial discussion about FLOSS in the specific context of SEE in Brazil and the benefits it may bring to the education of future Brazilian software engineers.

2. FLOSS and its development process

"FLOSS" is a broad acronym used for referring to a class of software projects that have Internet-based interaction between developers and public source code licensed under terms that comply to either the Free Software Definition by the Free Software Foundation[2] or the Open Source Definition by the Open Source Initiative[3]. Most software recognized as "free software" or "open source" complies with both definitions.

From a SE point of view, the most interesting aspect of FLOSS is its development process. A FLOSS project starts when an individual developer, or an organization, decides to make a software product publicly available on the Internet so that it can be freely used, modified, and redistributed.

After an initial version is released and advertised in the appropriate communication channels, early adopters start using the product. Some of these early adopters may be software developers themselves, who will be able to review the source code and propose changes that fix defects for their specific environments or add features for their own use cases. These changes may be sent back to the original developer(s) in the form of patches, which are files that contain only the difference between the original and modified versions. The project leader(s) will review the proposed changes and may apply them to the official version, so that when a new release is made, end users will have access to these new functionalities or bug fixes.

In the course of time, each release of the product may have more features and be more portable than its predecessor due to contributions from outside developers. The most frequent contributors may gain trust from the initial developer(s) and receive direct write access to the official source code of the project, becoming able to make changes directly to the official version of the product.

The development process is usually supported by a version control system (VCS), as illustrated in Figure 1. While the repository is often publicly available for read access, write access is restricted to a limited group of developers. Other developers will need their patches to be reviewed by a developer with the needed privileges to get their contributions into the project official repository.

The following characteristics make FLOSS projects different enough from "conventional" software projects, making the former interesting objects of study:

– Source code availability. Source code of FLOSS projects is always available on the Internet and most of the projects have a publicly-accessible version control repository.

[2] `http://www.gnu.org/philosophy/free-sw.html`
[3] `http://www.opensource.org/docs/definition.html`

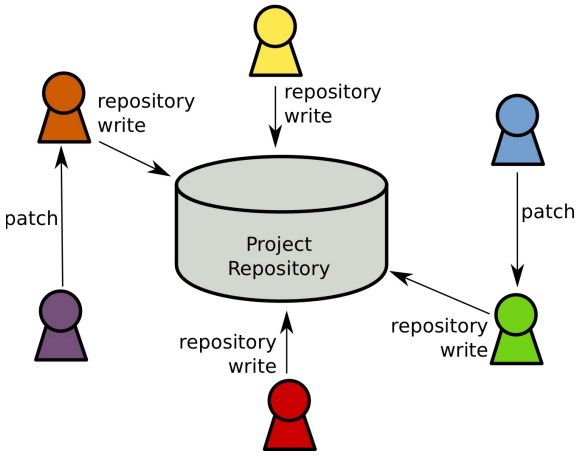

Figure 1. FLOSS development with a VCS repository.

– User/developer symbiosis. In most FLOSS projects the developers are also users of the software and they also provide requirements. Maybe because of that, several free software projects do not have explicit requirement documents and the development flows on a rhythm in which the developers are able to satisfy their own needs.

– Non-contractual work. A large amount of work in FLOSS projects is performed in a non-contractual fashion. This does not imply that the developers are necessarily volunteers, but only that there is no central management with control over all of the developers activities.

– Work is self-assigned. The absence of central management with control over the contributors activities promotes work self-assignment: volunteer developers will work on the parts of the project that most appeal to them and employed developers will work on the parts that are of most interest to their employers.

– Geographical Distribution. In most FLOSS projects, developers are spread among several different locations in the world. In projects with high geographical dispersion, communication is mostly performed through electronic means.

Although the SE literature tends to portrait FLOSS as a homogeneous phenomenon [Østerlie and Jaccheri 2007], most of these characteristics do not apply to all FLOSS projects and some of them may manifest in different ways across projects.

3. Opportunities in using FLOSS for Software Engineering Education

Software Engineering Education (SEE) has been supported by the efforts of several educational and scientific Computing societies and institutions. The Computing Curricula – Software Engineering Volume (CCSE) [IEEE and ACM 2004] that emerged from one of those efforts, provides guidance to academic institutions and accreditation agencies about what should constitute an undergraduate SEE.

Among other things, the CCSE states that exposure to aspects of professional practice of SE must be an integral component of the undergraduate curriculum, encompassing a wide range of issues and activities, including problem solving, management, ethical and legal concerns, as well as written and oral communication, working as part of a team and remaining current in a rapidly changing discipline [IEEE and ACM 2004].

Both FLOSS development and development processes taught in traditional SE textbooks address the challenges of multi-person construction and maintenance of multi-version programs [Parnas 1974], but with development processes, work practices, and project forms that differ significantly and in interesting ways [Scacchi 2010]. This poses the question of whether FLOSS development could be useful for educational purposes in SE undergraduate courses, specially for exposing students and educators to different aspects of professional practice.

The usage of FLOSS development in SEE has been exploited recently by quite a few forums, as discussed in Section 1. Most of the participants in those forums agree that undergraduate courses must involve students in large-scale software projects to expose them to real-world experience and understanding of the issues found in large, complex software projects, and that FLOSS projects can serve for this purpose and for other SEE challenges beyond that [Hislop et al. 2009].

Some opportunities brought up by the usage of FLOSS in SEE are [Ellis and Morelli 2008, Faber 2002, Hislop et al. 2009, Patterson 2006]:

– Working with "The Real Thing". In practice, undergraduate SE courses tend to be based on small projects that only require communication within each group of students rather than among groups. Working on FLOSS projects gives students the opportunity to work on real software projects, developing software that will actually be used by others, with distributed or co-located development in interaction with a community of developers and researchers from multiple countries.

– Learn by examining others' code. One of the most effective ways to learn coding and design techniques is working with an existing code base, so that the new developer is exposed to the work of other developers. In several successful FLOSS projects (e.g., Emacs, Linux, Apache), software is highly modular and the APIs are well documented. This way students can download the source code and documentation, as well as, begin to learn about specific modules and to code on specific parts of the project without the need to understand the entire design upfront. Students can also identify code smells and understand the trade-offs involved in removing them.

– Software Maintenance is the real issue. Most undergraduate SE courses tend to have the students developing projects from scratch. However, modifying existing code is a powerful way to learn good software development practices, as well as how to interact with other developers. Furthermore, the professional perspective of most graduates is more likely to be in the maintenance of existing software than in the creation of new software.

– Software quality exposed to the scrutiny of the world. Software engineers need to seek quantitative data on which to base decisions, yet also be able to function effectively in an environment of ambiguity and avoid the limitations both over-simplified, over-engineered or unverified modeling [IEEE and ACM 2004]. Student participation in FLOSS projects and communities offers excellent opportunities for performing software quality analysis over real source code and other artifacts that are made public by FLOSS projects. Just as conferences and journals promote peer-review of emerging ideas and outcomes, the FLOSS community performs similar type of scrutiny concerning software.

– Reinvigorating the curriculum and faculty members. FLOSS can provide a viable approach to a reinvigorated CS/SE curriculum and faculty members, needed for the benefit of future students in Computing. Patterson [Patterson 2006] suggests several approaches for dealing with FLOSS within courses, e.g., documentation, adding features, and removing bugs, given regular course and term constraints. Ideally, SE courses should include hands-on tasks and real-world scenarios. Faculty members that work only in an academic setting can gain practical experience in the disciplines they teach by joining a FLOSS project, where they can acquire practical experience in real development settings. Furthermore, FLOSS is project- and problem-based: developers work on projects that interest them and by working on interesting and meaningful projects they also learn correlative knowledge, skills, and aptitudes [Faber 2002].

– Contributing to relevant causes. FLOSS projects can mean more to students than "just" writing code: it may also support the seamless integration of topics such as health, sustainable development, and resource limitations into SE curricula [Ellis and Morelli 2008]. There is a large number of the so-called humanitarian FLOSS projects, which deal with issues like accessibility, medical record systems, public health monitoring systems, work coordination during disaster relief operations, and others. A reasonable number of these projects can be found in the website of The Humanitarian FOSS Project. During the Humanitarian FLOSS panel at the 6th International Conference on Open Source Systems (2010), it has been reported that working on such projects provides extra motivation for students, who feel they work can actually help make the world a better place.

– Reduced Costs. Although this is not the main point that FLOSS advocates would highlight, the "free as in *free lunch*" dimension of FLOSS also helps educational institutions with costs. Even though some tool vendors provide free proprietary software licenses for students, having a wide range of FLOSS available at no cost presents an advantage for faculty members. With FLOSS they can experiment with the tools without having costs for acquisition and the effort of getting a no-cost academic license for that expensive proprietary tool.

4. FLOSS-based SE courses

Involving computing students in FLOSS projects serves both the FLOSS community by providing development resources for the project and the academic community by providing access to large software projects in which students can gain experience. However, the same characteristics of FLOSS projects that make them useful for education can also make them difficult for both faculty members and students [Hislop et al. 2009]. Students and faculty must be familiar with these characteristics, the way FLOSS projects work in general and, if possible, the specific FLOSS projects in which students are going to work.

In this context, we provide a tentative list of topics that both students and faculty need to get acquainted with and that can be used in the design of FLOSS-based SE courses. Depending on the specific goals of the course, some items can be skipped, and some project-specific items can be added. These topics can also be spread throughout the curriculum and do not need to be addressed in a single course:

– Basic FLOSS Concepts. The GNU project, the Free Software Foundation (FSF), the Open Source Initiative (OSI), "The Cathedral and Bazaar" [Raymond 1999].

– FLOSS legal and economical aspects. Why share software: the concept of the commons, large-scale benefits. How to share: copyright legislation, copyleft, common, and recommended licenses, such as GPL, Apache, and MIT.

– The business of FLOSS. Business approaches, models, and opportunities with FLOSS projects.

– FLOSS social organization. How developers gain access and reputation in the projects. The concepts of meritocracy and "do-ocracy". Communication in FLOSS projects (mailing list, IRC, web forums, blog aggregators etc).

– FLOSS project management. Commonly used version control systems (SVN, Git, Mercurial, and Bazzar), bug tracking systems (Bugzilla, Redmine, etc). How to interact with the FLOSS community.

– Contributions. Performing translations, reporting bugs, and preparing and sending patches (features or bug fixes).

– Quality Assurance in FLOSS projects. Coding style, receiving and reviewing patches, automated tests, continuous build/integration, and bug triaging.

The above topics have been addressed in some of the undergraduate and graduate courses provided by UFBA and USP:

– Learning network operations and management (NOM). DCC-UFBA uses FLOSS exclusively on its servers since 2000. Technical activities are supervised by faculty and performed by students, who are encouraged to understand deeply the solutions maintained by them. Three former members of DCC-UFBA NOM team became official developers of the Foswiki FLOSS project, an important part of the department educational infrastructure. The user management application developed by the NOM team, that integrates several different services based on a centralized user database [Cason et al. 2007], has been published as a FLOSS project.

– Software reengineering. Another interesting initiative at DCC-UFBA was an advanced course for undergraduate students, with focus on software reengineering[4]. The contents were partially based on the open source book Object-Oriented Reengineering Patterns [Demeyer et al. 2003]. Reengineering patterns document knowledge about modifying legacy software, including principles, techniques, and skills applied in practice to help in diagnosing problems and finding appropriate solutions according to some reengineering goal [Demeyer et al. 2003]. Students were organized into 4 reengineering teams and played the role of practitioners who needed to reengineer existing open source applications, e.g., jGnash (a cross-platform personal finance application written in Java), Memoranda (a cross-platform diary manager and a tool for scheduling personal projects), and Gnometris (part of Gnome Games, a game derived from the classic falling-block Tetris). Since the use of version control systems is encouraged early in the undergraduate course, students were already able to work with at least one of them (Subversion). They had to perform comprehension activities with the support of open source tools, work with source code and bug repositories, perform static and dynamic analysis, extract design models, create new tests, etc. In short, the overall experience was very reinvigorating for students and the teacher as well. One student reported that the course was "the first course in which theory has been applied to a real problem, with no makeup".

[4] `http://disciplinas.dcc.ufba.br/MATB03/WebHome`

– Information, Communication, and Society of Knowledge. Between 2000 and 2008, professor Imre Simon, one of the most important Computer Science educators and researchers on Computer Science in Brazil, discussed the FLOSS phenomenon from a social, legal, and philosophical point of view in this undergraduate and graduate course at IME-USP. Simon adopted the best seller "The Wealth of Networks" [Benkler 2006] as the main text book and all content of this course was produced and released at a Wiki platform[5] on the Internet.

– Extreme Programming Laboratory. This course has used FLOSS projects as target to provide a practical and real development environment to study and apply Agile Methods [Goldman et al. 2004]. Since 2001, nearly 400 students from IME-USP that attended the course applied several techniques used in the FLOSS development process.

– FLOSS development. Since 2009, IME-USP provides a graduate course focused on FLOSS development[6], which is organized according to two goals. First, it aims at presenting to students, via books, journal, and conference publications, the state-of-the-art knowledge on FLOSS communities practices, theories to understand developers motivations and technology diffusion, the tools and repositories available, and, in general, what leads to success or failure in FLOSS development. In doing so, socio-technical, legal, managerial and philosophical issues are dealt with to create the basis to accomplish the second goal of the course, which is a FLOSS-related contribution from the students. Their contribution may take the form of a research or development project. When doing research, students write academic papers after planning data collection and analysis in such a way that their study can be added to the body of literature. In contrast, if a practical contribution is of their preference, they can find a FLOSS project and implement a feature requested by a user, for example, or create their own FLOSS project and, thus, release source code to attract external contributors and promote widespread adoption. The course finishes with a panel of experts evaluating student contributions to provide constructive feedback and help to define their grades.

5. Conclusions

Academia needs to recognize the value of FLOSS in teaching a quality SE program. In this paper, we presented our vision about the benefits that FLOSS can bring to SEE, suggested a list of topics that may be covered, and presented our experience with the usage of FLOSS in SE courses.

The benefits of engaging students in FLOSS projects do not come without challenges. Faculty must be prepared to assist students in adapting themselves to the project development environment, programming language, build tools, version control system, and even to non-technical aspects such as the existing project culture. That may bring a reasonable amount of effort, specially if faculty is not already familiar with FLOSS projects. Another challenge is matching participation in the projects with the actual course syllabus. It is not always easy to find opportunities for applying all the mandatory theoretical concepts on all projects, so updates on course curriculum may be required.

An important point to keep in mind is that student participation in FLOSS projects should also provide some benefit to the projects, in terms of contributions that can actually

[5] `http://conhecimento.incubadora.fapesp.br`

[6] `http://ccsl.ime.usp.br/wiki/index.php/MAC5856`

be used by others [Hislop et al. 2009]. When the existing project developers spend some of their time assisting and mentoring the students, useful contributions are the best way of paying them back.

Acknowledgements. The authors of this paper are supported by CNPQ, FAPESP, and the Qualipso project. This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES), CNPq grant 573964/2008-4.

References

Benkler, Y. (2006). *The Wealth of Networks: How Social Production Transforms Markets And Freedom*. Yale University Press.

Cason, D., Rocha, R., Terceiro, A., Barbosa, A., Ramos, E., and Galiza, H. (2007). Gerenciamento automático de usuários de uma rede acadêmica. In *Workshop Software Livre. Anais do Fórum Intern. Software Livre (FISL)*.

Demeyer, S., Ducasse, S., and Nierstrasz, O. (2003). *Object-Oriented Reengineering Patterns*. Morgan Kaufmann.

Ellis, H. and Morelli, R. (2008). Support for educating software engineers through humanitarian open source projects. In *Conf. on Software Engineering Education and Training Workshops*, pages 1–4, Los Alamitos, USA. IEEE Comp. Society.

Faber, B. D. (2002). Educational models and open source: resisting the proprietary university. In *Proc. of the 20th Annual Intern. Conf. on Computer Documentation*, SIGDOC'02, pages 31–38, New York, NY, USA. ACM.

Goldman, A., Kon, F., Silva, P., and Yoder, J. (2004). Being extreme in the classroom: Experiences teaching xp. *Journal of the Brazilian Computer Society*, 10:2004.

Hislop, G., Ellis, H., Tucker, A., and Dexter, S. (2009). Using open source software to engage students in computer science education. *SIGCSE Bull.*, 41:134–135.

IEEE and ACM (2004). Computing curricula – software engineering volume (CCSE). http://sites.computer.org/ccse.

Parnas, D. (1974). Software engineering or methods for the multi-person construction of multi-version programs. In Hackl, C., editor, *Programming Methodology*, volume 23 of *Lecture Notes in Computer Science*, pages 225–235. Springer.

Patterson, D. (2006). Computer science education in the 21st century. *Commun. ACM*, 49:27–30.

Raymond, E. S. (1999). *The Cathedral & the Bazaar* . O'Reilly & Associates, Inc., Sebastopol, CA, USA.

Scacchi, W. (2010). The future of research in free/open source software development. In *Proc. FSE/SDP Workshop on Future of Software Engineering Research*, FoSER'10, pages 315–320, New York, NY, USA. ACM.

Østerlie, T. and Jaccheri, L. (2007). A critical review of software engineering research on open source software development. In Wrycza, S., editor, *Proc. 2nd AIS SIGSAND European Symp. on Systems Analysis and Design*, pages 12–20. Gdansk Univ. Press.