Technology Innovation Management Review

A Model for Sustainable Student Involvement in Community Open Source

Chris Tyler

July 2009

"Open source better prepares students for the business world by exposing them to real-world problems and encouraging learning through the completion of real tasks. Open source amplifies a "hands-on" approach to learning by connecting students to a community of users in an effort to solve problems."

Jim Whitehurst, CEO of Red Hat

A healthy community is the lifeblood of any open source project. Many open source contributors first get involved while they are students, but this is almost always on their own time. At Seneca College we have developed an approach to sustainably involving students in open source communities that has proven successful in a course setting.

This paper outlines Seneca's approach and discusses the results that have been obtained with it. We examine the key factors for successful student integration into open source communities and steps that educational institutions and open source projects can each take to improve student involvement.

Barriers to Teaching Open Source Development

To effectively teach open source, it's necessary to move each student into the role of contributor. This appears straightforward, but ultimately proves to be an enormous challenge because: i) open source is as much a social movement as a technical one; and ii) many open source practices are the exact opposite of traditional development practices.

Many attempts to involve students in open source within a course have failed because everyone is overwhelmed:

1. The students, because they're suddenly facing an established codebase several orders of magnitude larger than any they have previously encountered in their courses, a community culture that they do not understand, and principles and ideals which are the opposite of what they've learned in other courses. For example, students are taught that answers and solutions should not be openly shared on the Web, that building on other's work by pasting it into your own is academically dishonest, and that it's wrong to deeply collaborate with peers on individual projects.
2. The professor and institution, because they're dealing with a continuously-changing, amorphous environment.
3. The open source project, because it is very difficult to deal with a sudden influx of students who tie up other contributors' time with questions and yet are unlikely to become long-term participants.

Distinctive Qualities of Open Source Development

In order to develop an effective approach to open source development, it's important to understand the qualities which make it unique:

1. Open source development is based around communities. These are generally much larger and more geographically diverse than closed-source development teams. They are enabled and empowered by the Web, leading to an increased focus on communication tools and internationalization and localization issues. Social issues become significant, and there is a productive tension between the need to maintain group discipline for coherence and the possibility of provoking a fork. Often, the culture of the community is not the culture of any particular member, but a synthetic intermediate culture.
2. The codebases managed by the larger communities range up to millions of lines in size and can date back many years or even decades. They often use tools and languages that are

About this article

Citation:

Tyler, C. 2009. A Model for Sustainable Student Involvement in Community Open Source. *Open Source Business Resource*, (July 2009). http://timreview.ca/article/272

Cite this article:

BibTex RTF Tagged MARC XML RIS

Author information

Chris Tyler
Seneca College

Chris Tyler is a programmer and Linux network administrator with a focus on the X Window System and LAMP. He has programmed in two dozen different languages over the past 20 years, and now teaches at Seneca College, Toronto.

More by this author

different from those taught in post secondary institutions, or employ common languages in unexpected ways. An example is using custom APIs that dwarf the language in which they are written, such as Mozilla's XPCOM and NSPR. These codebases require specialized, heavy-duty tools such as bug tracking systems, code search tools, version control systems, automated (and sometimes multi-platform) build and test farms and related waterfall and alert systems, toolchains for compiling and packaging each of the source languages used in the project, and release and distribution systems. Smaller open source projects which do not maintain their own infrastructure use some subset of these tools through a SourceForge account, Fedora Trac instance, or other mechanism.

3. Most open source systems have an organic architecture. Since it's impossible to anticipate the eventual interests and use-cases of the community at the inception of a project, the project requirements and development direction change over time. Although the lack of top-down design can be a disadvantage, the flexible, modular, and extensible architecture that often results has many benefits.

Each of these distinctive qualities presents a challenge to a traditional lecture-and-assignment or lecture-and-lab format course, but can be a strength in a community-immersed, project-oriented course. Carefully applied, these strengths can be used to overcome the barriers identified above.

Preparing to Teach Open Source

A prerequisite for teaching open source effectively is a professor who has one foot firmly planted in an open source community and the other in the educational world. To turn students into contributors, you need a dedicated conduit and liaison who can introduce students to the right people within the open source community.

On the academic side, the professor needs to connect with students on a personal level and to be aware of and able to navigate within the learning and administrative context of the educational institution. On the open source side, the professor must have deep contacts within the community, understand the community culture, and know what matters to the community so that projects selected for the students have traction. The professor must effectively use the community's tools and know when to use IRC, bugzilla, and email communications. The faculty member must adhere to open source principles and use the community's products in a production environment in order to have credibility.

The size of most large open source codebases prevents any one person from effectively knowing the entire codebase in detail, a problem that is compounded when multiple languages, layers, or major components are involved. One must move beyond being overwhelmed and become effective at searching, navigating, and reading code. The professor must demonstrate how to cope and show the students how to use community resources and contacts to find answers to questions. There is no textbook for this; it is behaviour that must be modeled.

Select an Open Source Community

An effective open source course requires the support of a large open source project. Project selection usually involves the faculty member(s) who will be teaching the course.

The open source community selected must have a sufficiently large scope to provide opportunities for many different types and levels of involvement. Its products must have many angles and components, so students can innovate in corners that aren't being touched by the main developers.

The reasons for selecting a larger community are straightforward:

1. A large community can absorb a large number of students spread across the various components and sub-projects within the community. This enables students with a broad range of interests and skills to get involved in a way that interests them. It also spreads student contact across the community, preventing overload of the existing contributors. Working within a single community provides a level of coherence for class discussions and for planning labs and lectures.

2. The project's infrastructure has usually scaled to the point where it will readily support the extra contributors.

3. Large projects tend to have broad industry support, opening up possibilities for spin-off research projects and broadening the value of the students' experience.

The key to project selection is to select something so big that the professor cannot directly manage the students and they are forced to interact with the community in order to succeed.

Q&A. Does the Google Summer of Code project provide any value to open source projects and the students who participate?

Google recently announced their fourth Summer of Code. Does the Summer of Code project provide any value to open source projects and the students who participate?

Lessons on Community Management from the Open Source World

From the outside (and often times from within, too), the success of healthy open source projects defies all logic. Scores of individuals from all over the world, all of whom have different skill levels, use cases, and experience, not to mention native languages and time zones, collaborate together in order to help make a project succeed.

How is it that all of this chaos comes together...

The Role of Consumers Within an Open Source Community

The software provided by the GNOME Project is produced by a large community comprised of several thousand developers, translators, quality assurance testers, and documentation writers. Consumers are represented in the community by technical users and organizations that distribute GNOME technologies. And while the community reaches out regularly to non-technical end users and welcomes any that...

Open Source as Community

What does it really mean to participate in an open source software community? If a company's open source strategy is limited to acting as end user of open source software, is there a business need to understand the nature of open source communities? Should it be the goal of all businesses to become an active participant in open source communities, or become recognized as a significant...

Select Potential Student Projects

Open source communities know what is interesting and valuable to them and are in the best position to suggest potential student projects. They're not always able to verbalize these projects, so the professor may need to suggest good ideas, but the community will recognize the value of ideas as they are proposed.

Some of the best project ideas are ones that existing community members would like to pursue, but can't due to a lack of available time or appropriate hardware. Project ideas should not be critical issues that directly affect release timelines or major community goals, but they may be of significant strategic value to the community. Each person proposing a project idea should be willing to be a resource contact for that project.

Potential projects can include a wide range of activities: feature development, bug fixing, performance testing, writing test cases, benchmarking, documenting, packaging, and developing or enhancing infrastructure tools.

The projects must then be screened for viability within the course context:

1. Are they the right size for the course? This does not mean that the project should be fully completed during the course. We look for projects that are not likely to be completed but which can be developed to a usable state in three months.
2. Are the necessary hardware and software resources available?
3. Is the level of expertise required appropriate for the type of student who will be taking the course? Ideally, each project should make the student reach high, but be neither stratospherically difficult nor trivially easy.

Prepare the Infrastructure

Each open source community has its own set of tools, and it's crucial that students use those tools so that community members can share with, guide, and encourage the new contributors. Existing community mailing lists, wikis, IRC channels, version control systems, and build infrastructure should be used by the students as they would by any other contributor.

Most academic institutions have their own computing and communications infrastructure. It's tempting to use these resources, but doing so draws a fatal line between the students and the rest of the community. Students can learn to use any tools, but the community will continue to use the tools they have established.

There's a certain amount of additional infrastructure needed to support an open source course, including:

1. A course wiki for schedules, learning materials, labs, project status information, and student details. If this wiki is compatible with the community's wiki, it will be easier for the community to contribute to learning materials.
2. An IRC channel set up in parallel to the community's developer channel(s), on the same network or server. These provide a safe place for students to ask questions which may provoke flaming in developer channels.
3. A planet to aggregate student blogs so community members can stay up-to-date. This should be separate from the community's main planet because some of the material will be course-specific.
4. Server farms and/or development workstations to ensure that the students have access to all relevant hardware and operating system platforms.

Teaching the Course

We start our open source courses by briefly teaching the history and philosophy of open source. We don't spend a lot of time on this topic because the philosophy will be explained and modeled in every aspect of the course.

Since open source is by its very nature open, we get students communicating immediately. They are required to establish a blog and submit a feed to the course planet. Almost all work is submitted by blogging, and students are expected to enter comments and to blog counterpoints to their colleagues' postings.

All course materials and labs are placed on the course wiki, and both students and community members are encouraged to expand, correct, and improve the material. These resources and the knowledge they represent grow over time and are not discarded at the end of each semester. This body of knowledge eventually becomes valuable to the entire community. Students are also required to get onto IRC. Since the main developers channels can be daunting, students are initially encouraged to lurk in those channels while communicating with classmates and faculty on the student channel.

At the start of the course, students begin reviewing the potential project list and are required to select a project by the third week. As part of the selection process, students will often use IRC or email to contact the community member who proposed a project that they are interested in. This is the first direct contact between the student and a community member, and since the student is expressing interest in something that the member proposed, the contact is usually welcome. It is critical that students choose projects that are important to the community and attract community support, so we prohibit them from proposing their own projects. Students often find it intimidating to select from the potential project list and the professor will often need to serve as a guide during project selection.

We prefer that each student select an individual project, with some rare two-person groups where warranted by the project's scope. Larger groups are almost always less successful. Students need to collaborate in the community instead of doing traditional, inward-focused academic group work. Students claim a specific project from the potential project list by moving it to the active project list and creating a project page within the course wiki.

Tools and Methodologies

Each community has a unique build process. This is often the first non-trivial, cross-platform build that students have encountered, so it's a significant learning experience that has a gratifying built-in reward. Students often go to great lengths testing different build options and approaches. Students also learn how to run multiple versions of the software for production and test purposes.

One of the challenges with building is finding an appropriate place to build, since many of the laptop computer models favoured by students have low CPU or memory, while student accounts on lab systems may not have sufficient disk space or student storage may be shared over a congested institutional network. Possible solutions include using external flash or disk drives with lab systems, or providing remote access to build systems.

As the students start work on their project, the course topics and labs teach the tools and methodologies used within the community. In most cases, the bug or issue tracking system drives the development, feature request, debugging, and review processes, providing an effective starting point. It's best that student projects have a bug/issue within the community tracking system, so students must either take on an existing bug or create a bug/issue for each project.

One useful exercise at this stage is to have the students "shadow" an active developer. On Bugzilla, a student can do this by entering that developer's e-mail address in their watch list, which forwards the student a copy of all bugmail sent to the developer. After coming to grips with the e-mail volume, students learn a lot about the lifecycle of a bug through this process.

Next, students need to learn how to: i) use code search tools such as LXR, MXR, and OpenGrok; ii) skim code; and iii) know who to talk to about specific pieces of code, including module and package owners and community experts. By working shoulder-to-shoulder with community members, particularly on IRC, they learn the ins-and-outs of the development process including productivity shortcuts and best practices. The professor can keep his finger on the pulse of the activity through IRC, guiding students when they get off track and connecting them with appropriate community members as challenges arise.

Students are expected to blog about their experiences on a regular basis, and all of the students and the community benefit from this shared knowledge. At the same time, differences between the student projects prevents one student from riding entirely on the coattails of other students.

Guest lectures by community developers have a powerful impact on students. Meeting a coding legend on IRC is great, but talking face-to-face and seeing a demonstration or hearing first-hand about the direction the software is headed has exceptional value.

We film these meetings and share the talks under open content licenses, making them available to people around the world. We've been surprised at the number of video views and by who is viewing them. We've found that new Mozilla employees often read our wiki and view the videos to help them come up to speed on the Mozilla codebase.

Releases and Contributions

Following the "release early, release often" mantra, students are required to make releases on a predetermined schedule. For the first open source course, three releases from 0.1 to 0.3 are required, and for the follow-on course, six biweekly releases from 0.4 to 1.0 are required.

We define the 0.3 release as "usable, even if not polished", reflecting the fact that a lot of open source software is used in production even before it reaches a 1.0 state. This means that the 0.3 release should be properly packaged, stable, and have basic documentation. It may be missing features, UI elegance, and comprehensive user documentation. The slower release rate in the first course is due to the initial learning curve and the fact that setting up a project and preparing an initial solution are time-consuming tasks.

As active members of an open source community, students are required to contribute to other open source projects, either those of other students or other members within the community. This contribution, which can take the form of code, test results, test cases, documentation, artwork, sample data files, or anything else useful to the project, accounts for a significant portion of the student's mark. Each project is expected to acknowledge external contributions on their wiki project page, and to welcome and actively solicit contributions from other students and community members. This requires that they make contribution easy, by producing quality code, making it available in convenient forms, and by explicitly blogging about what kind of contributions would be appreciated.

Students are often surprised to find community members contributing to their projects, but that is part of the authentic open source experience. It's important not to choke off collaboration for the sake of traditional academics.

In order to receive credit for contribution, students must blog about their contributions to other projects. At first this seems immodest to students, but the straight-facts reporting of work accomplished is a normal part of open development.

Seneca's Experience

Seneca College has been involved with open source for over 15 years, starting with Yggdrasil Linux installations in 1992. In 1999, we started a one-year intensive Linux system administration graduate program. In 2001, we introduced the Matrix server cluster and desktop installation, converting all of the hundreds of lab systems to a dual-boot configuration, which enabled us to teach the Linux platform and GNU development toolchain to students right from their first day at the college. In addition, a number of college faculty members released small open source software packages, including Nled, VNC#, and EZED.

In 2002, John Selmys started the annual Seneca Free Software and Open Source Symposium, which has since grown to a two-day event attracting participants from across North America.

In 2005, an industry-sponsored research project on advanced input devices created the need to modify a complex application. The lead researcher on this project, David Humphrey, contacted Mozilla to discuss the possibility of modifying Firefox. This contact led to a deep relationship between Mozilla and Seneca which outlasted that research project and led to the eventual development of the open source teaching model described here.

Our Open Source Development course implemented this model within the Mozilla community. David subsequently developed the Real World Mozilla seminar, which packs that course into an intensive one-week format, and a continuation course was eventually added to enable students to continue development on their open source projects and take them to a fully polished 1.0 release with faculty support.

Failures and Successes

The unpredictable nature of working within a functioning open source community poses peculiar challenges. We've had situations where a developer appears unexpectedly and posts a patch that fully completes a student's half-done project. Sometime students encounter reviewers who can't be bothered to do a review, stalling a student's work for weeks at a time, and some module and package owners have a complete lack of interest in the students' work. We've also had students drop the ball on high-profile work, or fail to grasp how to leverage the community and end up just annoying other contributors. In both cases our relationship with the community has taken a beating.

We've found that most students rise to the challenge presented in the open source development courses. Properly supported, students thrive when presented with big challenges. Conversely, coddling students in terms of project scope or expectations almost certainly leads to failure.

By and large, the open source development courses have been successful for the majority of students. Notable project successes include:

1. APNG: an extension of the PNG graphic format. While the PNG Development Group favored the use of MNG as the animated version of PNG, that standard proved difficult to implement effectively, and Mozilla wanted to try a lightweight, backward-compatible animated PNG format. Andrew Smith implemented APNG and his work has been incorporated into Firefox 3 and is also supported by Opera.

2. Buildbot integration: the Mozilla build system was adapted to work with the BuildBot automation system by Ben Hearsum.

3. Plugin-Watcher: many Firefox performance problems are believed to originate with third-party binary plugins such as media players and document viewers. Fima Kachinski, originally working with Brandon Collins, implemented an API to monitor plugin performance, and created a corresponding extension to provide a visual display of plugin load.

4. DistCC on Windows: DistCC is a distributed C compilation tool originally written to work with GCC. Tom Aratyn and Cesar Oliveira added support for Microsoft's MSVC compiler, allowing multi-machine builds in a Windows environment.

5. Automated localization build tool: there are many localizations that deviate in a minor way from another localization. Rueen Fiez, Vincent Lam, and Armen Zambrano developed a Python-based tool to apply a template to an existing localization to create the derivative version, which eliminates the need for extensive maintenance on the derivative.

In addition, a number of graduates are now employed full-time by Mozilla and companies involved in open source as a result of their work. The open source courses have also led to a number of funded research projects in collaboration with open source projects and companies.

Lessons Learned

There are many lessons which students repeatedly take away from the open source development courses:

- it's important to persevere
- it's acceptable to share and to copy code, within the context of the applicable licenses, instead

of guarding against plagiarizing or having your code stolen

- work in public instead of in secret
- tell the world about your mistakes instead of publicizing only your successes as there's a lot of value in knowing what does not work
- as a full community member you are a teacher as well as a student
- write down what you've done, and it will become a resource
- ask for help instead of figuring things out on your own
- key figures in open source are approachable, relationships are important and communication

is critical

- code is alive

We've also learned that open source is not for everyone. The least successful students do not engage the community and attempt to work by themselves. However, even students who don't continue working with open source take an understanding of open source into their career, along with an understanding of how to work at scale which is applicable even in closed-source projects.

Finally, we've learned that open source communities and companies have a huge appetite for people who know how to work within the community.

Where We're Headed

The open source courses are growing and will continue to work within the Mozilla project. In addition, we began working with OpenOffice.org in fall 2008. Our Linux system administration graduate program is being revised to incorporate many of the principles that we've used in the other open source courses. LUX students will be working directly with Fedoraproject.org, but on a much larger scale as LUX projects will span three courses across two semesters. A build automation course was introduced into our system administration and networking programs in January 2009. This course will also be based on work within the Fedora project.

In order to effectively leverage our open source teaching, research projects, and partnerships, we've created the Seneca Centre for the Development of Open Technology (CDOT) as an umbrella organization for this work.

Improving Student Involvement

Most open source communities actively welcome new contributors, but don't always make it easy to join. Steps a project can take to encourage contributors will improve student involvement:

1. Make it easy for new contributors to set up your build environment. Create an installable kit of build dependencies, generate a metapackage, or provide a single web page with links to all of the required pieces.
2. Create a central web page with links to basic information about your project that a new contributor will need, such as build instructions, communication systems, a list of module owners, a glossary or lexicon of community-specific technical terms and idioms, and diagrams of the software layers and components used in your products.
3. Create sheltered places or processes to enable new people to introduce themselves and get up to speed before being exposed to the full flaming blowtorch of the developer's lists and channels. This might include an e-mail list for new contributors, self-introductions, or an IRC channel for new developers.

In addition, in a course context:

1. Ensure that the community is aware of the course and course resources.
2. Feel free to join the student IRC channel, contribute to student projects as you would any other project, and read the student planet.
3. Contribute to learning materials on the course wiki.
4. Apart from recognizing the students as new community members, treat them as any other contributor.

Conclusion

Open source development is dramatically different from other types of software development, and it requires some radically different pedagogical approaches. A community-immersed, fully-open, project-oriented approach led by professor who is also a member of the open source community provides a solid foundation for long-term, sustainable student involvement in that community.

This article is based on a paper that was presented at Linux Symposium 2008 and published in the 2008 Proceedings, Volume 2. A copy of the original paper, along with related resources, is available from the author's website.

 ISSN: 1927-0321 Formerly the *Open Source Business Resource*