A Model for Sustainable Student Involvement in Community Open Source

Chris Tyler
Seneca College
chris.tyler@senecac.on.ca

Abstract

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. I will 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.

1 The Challenge

To effectively teach Open Source, it's necessary to move each *student* into the role of *contributor*. At first blush this appears straightforward, but it ultimately proves to be an enormous challenge because Open Source is as much a social movement as a technical one and because many Open Source practices are the exact opposite of traditional development practices.

1.1 Barriers to Teaching Open Source Development

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

- 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, that answers and solutions should not be openly shared on the web [15], 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.
- The professor and institution, because they're dealing with a continuously-changing, amorphous environment.
- 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.

1.2 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:

- Open Source development is based around *communities*. These are generally much larger and more geographically diverse than closed-source development teams, and 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.
- The *codebases* managed by the larger communities range up to millions of lines in size and can

date back many years or even decades. Furthermore, they often use tools and languages that are different from those taught in post secondary institutions, or employ common languages in unexpected ways—for example, 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 [20], fedorahosted.org Trac instance [8], or other mechanism.

- Most Open Source systems have an *organic architecture*. Since it's impossible to anticipate the eventual interests and use-cases of the community—including downstream needs—at the inception of a project, the project requirements and development direction change over time and the project grows into its final form (I've never seen UML for an Open Source project!). Although the lack of top-down design can be a disadvantage, the flexible, modular, and extensible architecture that often results has many benefits.

1.3 Turning Challenges into Strengths

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.

2 Preparing to Teach Open Source

2.1 Select a Faculty Member

A prerequisite for teaching Open Source effectively is a professor who has one foot firmly planted in the Open Source community and the other in the educational world. In order 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 (and friendships!) within the community, understand the community culture, and know what matters to the community so that projects selected for the students have traction. She must also know and effectively use the community's tools—for example, knowing when to use IRC (Internet Relay Chat), when to use Bugzilla, and when to use e-mail to communicate. The faculty member must have bought-in to Open Source principles, and use the community's products in a production environment ("eating your own dogfood")—there's no credibility to lecturing about bugzilla.mozilla.org using Safari, or presenting PowerPoint slides about OpenOffice.org.

The massive size of most large Open Source codebases prevent 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. This leads to the need to be *productively lost* in the code—moving beyond being overwhelmed and becoming effective at searching, navigating, and reading code. The professor must demonstrate how to cope in this state instead of pretending to know each line, and this includes pulling back the curtain and showing the students how she uses community resources and contacts to find answers to questions. There is no textbook for this; it is behavior that must be modeled.

2.2 Select an Open Source Community

An effective Open Source course also requires the support of a large Open Source project. This selection of project is usually informed by the involvements of 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 also have many angles and components, so students can innovate in corners that aren't being touched by the mainline developers. This likely narrows the list

of potential candidates down to the top one hundred or so Open Source projects, which includes the major desktop applications, graphical desktop environments, key server applications, kernels, and community-based Linux distributions.

The reasons for selecting a larger community are straightforward:

- 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, using the Open Source model of having people work on things they are passionate about. It also spreads the student contact across the community so that few developers will have direct contact with more than one or two students, preventing overload of the existing contributors. At the same time, working within a single community provides a level of coherence that makes it much easier to hold class discussions and plan labs and lectures than if the students' involvement was spread across a number of smaller, independent Open Source communities.

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

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

To make this work, you will need the support of the community; they must buy into the idea of teaching students to become productive contributors—not a hard sell, because most communities are hungry for contributors—and there must be open lines of communication with the community's leaders.

It is counter-intuitive to select a large community because it seems easier to manage the students' involvement in a smaller project—but the key 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.

2.3 Select Potential Student Projects

Open Source communities know what is interesting and valuable within their own space 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 poke and prod to shake out good ideas, but the community will recognize the value of ideas as they are proposed.

Some of the best project ideas as ones that existing community members would like to pursue, but can't due to a lack of available time (or in some cases, a lack of appropriate hardware). These issues should not be blocker bugs or critical issues that will 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, performing 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:

- Are they the right size for the course? This does not mean that the project should be fully completed during the course; we've taken an idea from Open Source—the "dot release"—to replace the idea of "complete work," and we look for projects that are not likely to be completed but which can be developed to a usable state in three months.

- Are the necessary hardware and software resources available?

- 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.

2.4 Prepare the Infrastructure

Each Open Source community has its own set of tools, and it's crucial that students use those native tools so that

community members can share with, guide, and encourage the new contributors. The 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, including tools such as Moodle, Blackboard, version control systems, instant messaging systems, forums and bulletin boards, and so forth. It's tempting to use these resources because they are familiar and to avoid placing a burden on the community's 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; when the students meet them there, as fellow contributors, great interaction takes place.

However, there's a certain amount of additional infrastructure needed to support an Open Source course, including:

- A course wiki for schedules, learning materials, labs, project status information, and student details. If this wiki is compatible with the community's wiki (using the same software and similar navigation), it will be easier for the community to contribute to learning materials.
- An IRC channel set up in parallel to the community's developer channel(s), on the same network or server. We have established #seneca channels on irc.freenode.net and irc.mozilla.org, for example; these provide a safe place for students to ask the sorts of questions which may provoke intense flaming in developer channels.
- A blog planet to aggregate the student's blog postings so that all community members, including the students themselves, can easily stay up-to-date on what all of the students are doing. This should be separate from the community's main planet because some of the material will be course-specific. (It's a good idea for the professor to feed the community planet to keep the community up-to-date with what the students are doing.)
- Server farms and/or development workstations (as appropriate to the projects undertaken), to ensure that the students have access to all relevant hardware and operating system platforms.

3 Teaching the Course

We start our Open Source courses by briefly teaching the students the history and philosophy of Open Source. We do this using classic resources such as *The Cathedral and the Bazaar* [24] and the film *Revolution OS* [23], but 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.

3.1 Communication

Since Open Source is by its very nature *open*, we get students communicating immediately so that they get used to working in the open. They are required to establish a blog (on their own website, or on any of the blogging services such as blogger.com or livejournal) 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 to use, students are initially encouraged to lurk in those channels while communicating with classmates and faculty on the student channel. The parallel channel enables students (and faculty) to provide commentary on #developers chatter in real time without annoying the developers, and it provides an appropriate context for course-related discussion. Since the student channel is on the same network/server as the developers' channels, some existing community developers will join the student channel.

3.2 Project Selection

At the very 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 e-mail 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 do find it intimidating to select from the potential project list, since the things that matter to the community are big, hard, and mysterious (or at least appear that way). The professor will often need to serve as a guide during project selection.

We strongly prefer that each student select an individual project, with some rare two-person groups where warranted by the project scope; larger groups are almost always less successful. Students need to collaborate in the community—both inside the class community and within the larger Open Source 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.

3.3 Learning How to Build

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, and one that has a gratifying built-in reward. There's a lot of easy experimentation available here, so students often go to great lengths testing different build options and approaches (discovering, for example, that a particular build takes 8 or more hours on an Windows XP system, but only about 40 minutes on a Linux VM under that same XP system). The 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 favored by students may have low CPU "horsepower" 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.

3.4 Tools and Methodologies

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 (such as Bugzilla) 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 [4], which forwards to 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, the students need to learn how to cope with being *productively lost* by using code search tools (such as LXR [9], MXR [10], and OpenGrok [11]), learning to skim code, and most importantly, learning 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.

As with all of the activity in the course, students are expected to blog about their experiences on a regular basis, and all of the students benefit from this shared knowledge (as does the community, which does not have to answer the same questions over and over again). At the same time, differences between the student projects prevents one student from riding entirely on the coattails of other students.

3.5 Meeting the Community

Guest lectures by community developers have an enormously powerful impact on students: meeting a coding legend on IRC is great, but talking to him face-to-face and seeing a demonstration of how he works 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 views these videos have received, and who is viewing them: for example, we've found that new Mozilla employees often read our wiki and view the videos of our Mozilla developer talks as they come up to speed on the Mozilla codebase.

3.6 Releases

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.

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, although 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.

3.7 Contribution to Other Projects

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 in turn 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 (and community members are sometimes unsure whether doing so is permissible from an academic point of view), 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.

4 Seneca's Experience

4.1 History

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 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* [25], *VNC#* [22], and *EZED* [21].

In 2002, John Selmys started the annual Seneca Free Software and Open Source Symposium [17], which has since has 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.

Seneca College's DPS909/OSD600 *Open Source Development* course [19] 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 the DPS911/OSD700 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.

4.2 Failures

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.

On the other hand, 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 to them in the Open Source development courses. This has meant that, properly supported, students thrive when presented with big challenges. Conversely, trying to protect students by coddling them in terms of project scope or expectations ("throwing them into the shallow end of the pool") almost certainly leads to failure.

4.3 Successes

By and large, the Open Source Development courses have been successful for the majority of students. Notable projects successes by Seneca students include:

- APNG [1] – Animated PNG format, an extension of the PNG [14] high-colour-depth, full-alpha graphic format. While the PNG Development Group favored the use of MNG as the animated version of PNG, that standard had proven to be large and difficult to implement effectively, and Mozilla wanted to try a lightweight, backward-compatible animated PNG format. Andrew Smith implemented this format [2] and his work has been incorporated into Firefox 3; Opera now also supports APNG.
- Buildbot integration – The Mozilla build system was adapted to work with the BuildBot automation system by Ben Hearsum [6].
- Plugin-Watcher – Many Firefox performance problems are believed to originate with 3rd-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 [13].
- DistCC on Windows – 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 very minor way from another localization (for example, en_US and en_CA). Rueen Fiez, Vincent Lam, and Armen Zambrano developed a Python-based tool that will apply a template to an existing localization to create the derivative version, which eliminates the need for extensive maintenance on the derivative [3].

In addition, 4 out of 25 student interns at Mozilla this summer are from our courses, and 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.

4.4 What We've Learned

There are many lessons which students repeatedly take away from the Open Source development courses:

- It's important to persevere.
- It's OK to share and to copy code (within the context of the applicable Open Source 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—there's a lot of value in knowing what does not work.
- You are a full community member, which makes you a teacher as well as a student. Write down what you've done, and it will become a resource. (It's interesting to note that many of the Google searches which the students are performing now return our own course wiki and blogs.)
- Ask for help instead of figuring things out on your own.
- Key figures in this industry do not stand on pedestals—they are real people and are approachable. Relationships are important and communication is critical.
- Code is alive.

We've also learned that Open Source is definitely not for everyone. The least successful students are those who do not engage the community and who attempt to work strictly 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.

4.5 Where We're Headed

The OSD/DPS courses are growing and will continue to work within the Mozilla project. In addition, we will also be working with OpenOffice.org [12] this fall.

Our Linux system administration graduate program (LUX [18]) 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 the Fedora project [7], but on a much larger scale than the Mozilla and OpenOffice.org projects: LUX projects will span three courses across two semesters.

One other course is in development: a build automation course, scheduled to be 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) [16] as an umbrella organization for this work.

5 Steps an Open Source Community Can Take to Improve Student Involvement

Most Open Source communities actively welcome new contributors, but don't always make it easy to join. Many of the steps a project will take to encourage contributors of any sort will improve student involvement:

- 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.
- 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. It's challenging for new contributors to even map IRC nicks to e-mail addresses and blog identities!
- 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-contributor self-introductions or a process for self-introduction on the main lists, or an IRC channel for new developers.

In addition, in a course context:

- Ensure that the community is aware of the course and course resources.

- Feel free to join the student IRC channel, contribute to student projects as you would any other project, and read the student planet.
- Contribute to learning materials on the course wiki.
- Apart from recognizing the students as new community members, treat them as any other contributor.

6 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 Open Source community.

7 Acknowledgments

I would like to acknowledge the pioneering work of my colleague David Humphrey in establishing the Open Source Development courses at Seneca, and for his thoughtful review of this paper.

References

[1] *Animated PNG Information Site*. http://animatedpng.com/.

[2] *APNG project page*. http://zenit.senecac.on.ca/wiki/index.php/APNG.

[3] *Automated Localization Build Tool project page*. http://zenit.senecac.on.ca/wiki/index.php/Automated_localization_build_tool.

[4] *Bugzilla Watch Lists*. http://www.bugzilla.org/docs/3.0/html/userpreferences.html#emailpreferences.

[5] *DiscCC with MSVC project page*. http://zenit.senecac.on.ca/wiki/index.php/Distcc_With_MSVC.

[6] *Extending the Buildbot project page*. http://zenit.senecac.on.ca/wiki/index.php/Extending_the_Buildbot.

[7] *Fedora Project*. http://fedoraproject.org/.

[8] *Fedorahosted Trac instances*. https://fedorahosted.org/web/.

[9] *LXR*. http://lxr.linux.no/.

[10] *MXR*. http://mxr.mozilla.org/.

[11] *OpenGrok*. http://opensolaris.org/os/project/opengrok/.

[12] *OpenOffice.org*. http://openoffice.org/.

[13] *Plugin-watcher project page*. http://zenit.senecac.on.ca/wiki/index.php/Plugin-watcher.

[14] *PNG - Portable Network Graphics*. http://www.libpng.org/pub/png/.

[15] *The Ryerson Facebook Dilemma*. http://www.wikinomics.com/blog/index.php/2008/03/12/the-ryerson-facebook-dilemma/.

[16] *Seneca Centre for Development of Open Technology (CDOT)*. http://cdot.senecac.on.ca/.

[17] *Seneca Free Software and Open Source Symposium*. http://fsoss.senecac.on.ca/.

[18] *Seneca LUX Graduate Program*. http://cs.senecac.on.ca/?page=LUX_Overview.

[19] *Seneca Open Source Development Wiki*. http://zenit.senecac.on.ca/wiki/.

[20] *Sourceforge*. http://sourceforge.net/.

[21] John Flores. *EZED - Easy Editor*. http://cdot.senecac.on.ca/software/ezed/.

[22] David Humphrey. *VNC#*. http://cdot.senecac.on.ca/projects/vncsharp/.

[23] J. T. S. Moore. *Revolution OS*, 2001. http://www.revolution-os.com/ (available online at http://video.google.com/videoplay?docid=7707585592627775409).

[24] Eric Raymond. *The Cathederal and the Bazaar*, 2000. http://catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/.

[25] Evan Weaver. *NLED—Nifty Little Editor*. http://cdot.senecac.on.ca/software/nled/.