Is There a Role for Open Source Software in Systems Analysis?

Michael P. Conlon
michael.conlon@sru.edu

Frank W. Hulick
frank.hulick@sru.edu
Computer Science Department, Slippery Rock University of Pennsylvania
Slippery Rock, PA 16057, U.S.A.

Abstract

Open source software has enjoyed considerable success in recent years, as measured by the growth both in its popularity and in the number and complexity of available programs. However, there is little mention of open source software in today's systems analysis textbooks. This paper explores the role that open source software should play in systems analysis, and in the systems analysis course.

Keywords: systems analysis, open source software, free software, software development

1. INTRODUCTION

Free/Open Source Software

In 1999, the term *open source* was first applied to what had been called *free software*. Several participants in the free software movement realized that the multiple denotations of the word *free* were causing confusion among potential users of free software. In particular, free software was being underutilized in the commercial arena because of managers' belief that free software must be valueless software, i.e., you get only what you pay for. However, the word *free* here refers to freedom, not price. Richard Stallman, a founder of the free software movement, defines *free software* in the form of four freedoms (Free Software Foundation, 2005):

- The freedom to run a program, for any purpose.
- The freedom to study how the program works, and adapt it to your needs. Access to source code is a precondition for this criterion.
- The freedom to distribute copies.
- The freedom to improve the program and release your improvements to the public, so that the whole community benefits. Access to the source code is a precondition for this criterion as well.

Clearly, these freedoms are important to commercial users as well as hobbyists and academics. By emphasizing the availability of source code, we sidestep the *libre/gratis* confusion. Use of the phrase *open source software* is not intended to de-emphasize the importance of freedom, but rather to eliminate the popular confusion.

Roots of Open Source Software

Open Source software is not new. It has its roots in the user groups of the major computer hardware vendors and in the computer science laboratories of universities, where a culture of sharing software has prospered. It is important to

realize that proprietary software is, in fact, newer than open source software, and that proprietary software vendors actually needed to convince the programmer community that software sharing should not be the norm. Bill Gates, in "An Open Letter to Hobbyists," protested that his software was not to be shared (Gates, 1976). The success of the personal computer revolution, and Microsoft's concomitant rise, led to the general perception that proprietary, closed source software should be the norm. In the Unix community, development and use of open source software continued, but these efforts did not initially attain wide recognition because of the failure of the Unix vendors to penetrate the personal computer market.

The virtually complete lack of marketing and advertising effort associated with open source software permits a general ignorance of the very existence of this segment of the software world. Similarly, many who have heard of open source software have the mistaken impression that its impact is negligible. In fact, there are many successful open source programs. Foremost among these are the programs that were running the Internet before proprietary Internet software was created. Among the more-significant open-source programs are

- *Routed, Bind, Sendmail*, and *Apache,* which provide Internet routing, name service, e-mail transfer, and Web service
- *Linux, OpenBSD, NetBSD, FreeBSD, and FreeDOS* operating systems
- The Gnu Compiler Collection (gcc) and the Gnu utilities
- *Samba*, which provides file and printer sharing services simulating a Windows server
- *MySQL* and *PostgreSQL* database management systems
- *OpenOffice.org* office suite
- *Mozilla* and *Firefox* Web browsers
- The *KDE* and *Gnome* desktop environments, each of which provides a plethora of application programs, from editors and utilities to finance managers and multimedia applications.

Motivation for Including Open Source in Systems Analysis Curricula

Since open source clearly represents a significant segment of the software world, it deserves consideration in systems analysis courses and textbooks. One might ask why it is not discussed there already.

This situation can be explained in that, historically, open source software tended to be systems software, not the usual domain of systems analysis. However, as the systems software has stabilized, open-source programmers are moving more and more into application programming. As businesses perceive advantages in open source development, they will need more systems analysts who understand open source development processes.

2. OPEN SOURCE IN THE SOFTWARE DEVELOPMENT LIFE CYCLE

Build or Buy?

Often, systems analysis is performed in order to specify software for acquisition rather than for development. An advantage of off-the-shelf software is reduced risk, since, before the firm commits to it, the software is known to work. A disadvantage is that the software may not be a good fit for the firm, and the firm might need to make inconvenient changes to its business processes, and perhaps write custom workaround software to accommodate the acquired software to the firm's legacy systems.

Open Source Reduces Risk

Perhaps the simplest way to include open source into systems analysis is to consider existing open source software as well as proprietary software when making the "build-or-buy" decision, which now becomes the "build, buy, or download" decision. Appropriate open source software gives us the best of both building and buying. Risk is reduced because the software is known to work (and it can even be tested before any commitment to it is made), and, because of the availability of source code, the program can be customized to the firm's specific needs.

Risk is further reduced when open source software is chosen, because open source

software is written to community standards. There are no secret, proprietary file formats or secret communication protocols in open source software, since it is not to the advantage of anyone writing open source software to foster user lock-in. This means that the firm's data will be accessible well into the future. Even if standards change, open standards are well-documented, so any competent programmer can write a program to convert data to any new format.

Additional risk reduction comes from the very openness of the code. Since anyone can see the code, there is little chance that a security trapdoor can be introduced undetected. Additionally, because the firm possesses the product's source code, there is no danger of the product's discontinuance because of a vendor's merger, bankruptcy, or change in marketing strategy. If the software is useful to the firm, the firm can continue to use, maintain, and extend it.

Commercial Open Source Development

Another option for the firm is to build software rather than to acquire it. Should open source development be considered? Isn't it folly for a company to give away the results of its efforts? The answer depends on the firm's business model.

A company that makes most of its money by licensing software would be foolish to donate its software to the open source world, unless it is planning to change its business model. Don't expect *Microsoft Word, WordPerfect, or Quicken* to become open source anytime soon.

However, most programmers and analysts are not employed by companies that license software. They are employed by companies that use software (Raymond, 1999). There are distinct advantages to such a company in open-sourcing its software products. Their small IT staff may be overworked, but if their software is useful to other companies, those companies' programmers may contribute to the software project. This effectively extends the company's development staff without extending its payroll. The resulting independent peer review of the software can facilitate the development of more-reliable, feature-rich software for less cost.

There are even reasons for software-for-licensing companies to consider going open source. Often there is more money to be made in supporting software users than in selling software licenses. By open-sourcing a product, a company might develop a larger market, and the support business could be lucrative. Red Hat Linux and MySQL are products of such (profitable!) companies.

Open Source Methodology

Many open source projects are organized with a single leader or a small leadership committee (simplified to just *leader* henceforth). The leader decides whether to adopt any proposed software change, the sole criterion being the technical merit of the proposed change. Since the code base of an open source project is placed in a public repository, such that anyone can download, view, and modify the source code, anyone at all can suggest any change whatsoever. So, what constitutes *technical merit*?

In proprietary software development it is expected that documents have been developed which specify the scope of the project, its financial feasibility, and a schedule for its completion. Code is developed in accordance with the planning documents, so there is no question about the code's merit. It would be rather unusual for a programmer involved in proprietary development to contribute a feature outside the scope of the plan. However, development projects have been known to fail in spite of such planning.

Open source projects generally do not have such planning documents, yet "bad" code gets rejected and "good" code gets accepted. Ultimately, it is the team of developers on the project who determine what constitutes "good" code. These developers have self-selected themselves for the project, so they embody a good deal of domain expertise (Morton, 2004). If the leader says certain code is bad, s/he can expect considerable opposition from the team if they disagree. A leader who disregards the opinions of the team risks losing leadership. Open source projects have acquired an excellent record for quality, so the open-source quality-assurance process certainly works.

Among the advantages of open-source development is its resistance to externally-

mandated scope creep, which is often cited as one of the major causes of project failures. Because the people determining technical merit in open-source projects are developers and not managers, they tend to accept changes based on the practicality and usefulness of the changes, rather than on criteria related to marketing, or to someone's status within the corporate hierarchy. Sometimes, external developers may contribute features outside the defined scope of the project. Since they have taken it upon themselves to design the new feature, it represents no cost to the firm, and since it comes from someone with genuine concern for the project, it may well be that this new feature belongs in the system in spite of its omission from design documents.

One type of planning not found in successful open source projects is schedule planning. While deadlines exist, they are set by the development team rather than managers, and they are not set until it becomes apparent to the leader that the current phase of development is nearing completion. Since no one ever really can tell how long something that has never been done before can take, the real purpose of deadlines is to establish a limit on development time. A firm that does open source development will have to be content with not knowing very far in advance when their project will be completed. In reality, it is *never* possible to know the completion time in advance, but in open source development there is no attempt to pretend otherwise. When the developers set the deadlines, software is released when it is ready, with minimal bugs. If someone really needs the software before it's ready, they can always download it from the code repository. Research even seems to indicate that this lack of scheduling actually results in the fastest delivery of a working system (DeMarco and Lister, 1987).

None of this means that open source projects don't fail. Browsing through the open source projects at sourceforge.org will reveal many projects that are inactive. Projects may become inactive for many reasons, other than successful completion: The leader lost interest and never attracted a community of developers to take over; the project wasn't carefully thought-out and never made significant progress; the project was not feasible; the project duplicates another successful project. The good news is that someone else attempted these failed projects, so your firm's resources were not wasted in the process.

There is some question as to what problems are appropriate for open source development. Andrew Morton, one of the leaders of the Linux project, has suggested that good open source projects deal with problem domains which are well-understood, such as operating systems, compilers, Internet infrastructure, databases, word processors, and the like (Morton, 2004). Eric Raymond (Raymond, 1999) agrees. When a firm attempts a state-of-the-art software project, it may not find a community of programmers who understand the problem, thus bearing much of the cost of development itself. It would be difficult for such a firm to justify donating such a project. One would need to question whether, as the project progresses, it will collect an external following to contribute to further development, and whether the benefits of such contributions would be preferable to the income obtainable from licensing the program.

Initiating Open Source Development

A firm should start its open source software project in much the same way as if it were not open source. Determination of business requirements and the technical feasibility study are as important as ever. Check for related government or community document-format or communication-protocol standards. If such standards exist, conformance with these standards must be specified.

From his experience in the f*etchmail* experiment, Eric Raymond (1997) suggests that the next step is for the firm to look for an open-source project that approximates its requirements. This eliminates some risk: the starting code, however incomplete, still works.

Assume, as happened with *fetchmail*, that such a project exists, with some working code, but that many or even most requirements are not met. By contributing improvements to this code, the firm’s programmers will start to get feedback from the leader and other members of the team.

As more and more improvements are contributed, and if the contributions are constructive, the firm's programmers will become trusted within the team. This will lead to their being given write-access to the code repository. One of them may even be asked to take over leadership should the current leader have lost interest in the project.

After some time the firm will come to one of two conclusions: either this software is going to solve the problem, or, as happened in the *fetchmail* experiment, a complete re-write is necessary. In the former case, the firm needs only to proceed as it is already. In the latter case, it has, in effect, refined the problem, and is now prepared to re-write the specifications and structural design. This is not a failure: the firm has just avoided the "This is what we asked for but this is not what we need" problem. Brooks wrote, "Plan to throw one away; you will, anyhow." (Brooks, 1995)

If there seems to be no open-source program that approximates the firm's needs, this is the point where development starts.

Raymond insists that a project cannot begin in *bazaar style*, i.e., with large numbers of geographically dispersed, self selected team members. On the other hand, a polished, final product isn't necessary, either, before soliciting outside developers. What is needed is a program which can "(a) run, and (b) convince potential co-developers that it can be evolved into something really neat in the foreseeable future" (Raymond, 1997), even if the firm must create that much itself.

Once again, Raymond's *fetchmail* project serves as a model for development. As the replacement system is designed and built, the code should be posted on the Internet for public access. Postings must occur regularly; waiting until the code is perfect would be a mistake. Of course, suitable disclaimers about the stability of the code should be posted, too. If the project is useful, developers from the old project will be attracted, and new ones as well. These people will help find bugs and contribute fixes and improvements, and the system will approach stability rapidly. Raymond states, "Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging (Raymond, 1997)." Note that this procedure has a lot in common with *Extreme Programming* (Beck, 1999).

Thus, testing is integrated with development: the openness of the code means that people will try the code, well before it's ready for final release. Bugs that would not have been noticed become apparent to *someone* in the mass of users trying out the system. Linus' Law applies: "Given enough eyeballs, all bugs are shallow" (Raymond, 1997). A major feature of open source development is that it bypasses Brooks' Law, multiplying the ability to find and fix bugs.

The pervasiveness of the Internet is the single development which has catapulted open source development to the fore. Programmers will be productively developing software with team members who have never met each other before. Most communication within the design team in open source development occurs over e-mail and, to a much lesser extent, IRC (Internet Relay Chat) (Morton, 2004). Unlike with proprietary development, all of the design conversations (and disagreements) are public. Expect that "dirty laundry" will be hanging out; this is a requirement for a democratic process. Mailing lists are archived, providing a running record of design conversations and decisions. If you are disturbed by the frank, public discussions related to the system you are developing, remember that proprietary development has lots of dirty laundry, too, but the public is rarely privy to the conversations.

3. CHANGES TO THE SYSTEMS ANALYSIS COURSE

The above discussion necessitates the following changes in systems analysis course content:

- Add open source into the menu of options in the (renamed) "build, buy, or download" decision. Treat the open source option as a low-risk option, explaining that the low risk derives from the facts that the code is known to run, the code can be modified to meet a firm's specific needs, and that the code will not be a captive of a vendor's insolvency, acquisition, or changes in

market strategy.

- Add *download-and-modify* as an option intermediate to *build* and *buy*. This is an option that was previously unavailable, and it gives the firm significant flexibility. Point out that the download option offers the advantage of giving code customized to the firm's needs without the need for the firm to bear all of the development costs itself.
- Discuss open source development as a valid option when the decision is made to develop custom software. Among the reasons for developing new software under an open-source regimen are the potential assistance from outside developers, which leads to rich functionality and minimal bugs, and the distribution of development costs across all the firms that take an interest in the software.
- Point out the advantages of giving software away to the community, as well as the circumstances when proprietary development makes better sense. These advantages include the software improvements discussed in the preceding point, and the potential income from selling support. Open-sourcing software may facilitate its wider distribution, thus giving the firm greater potential for income from support contracts. Indicate that keeping the source code closed makes most sense when the code embodies trade secrets or when the firm expects to make significant income from licensing the software.
- Emphasize the importance of the openness of the process when open source development is chosen, since outside contributors will not join a partially-closed process. Discuss how open source development trades control for outside assistance. Tell students that an open process means public discussions, and even arguments, about design decisions. This is necessary to achieve the best possible technical solution.
- Stress that release of open source software must be both early and often, at least in the early phases of development. Regular, frequent releases encourage the developer community, tempting them to try out the latest version and return bug reports and fixes, and serve as an incentive for them to get involved.
- Point out that this outside assistance can both help eliminate bugs and drive faster development.
- Finally, point out that open-sourcing software is not a panacea. A project that is not well-thought-out and competently led will fail, whether the development process is open or closed.

4. CONCLUSION

Open source software has become an important part of the software world. It makes economic sense for many development projects. Systems analysts and designers need to understand its economics and peculiar development processes. It is incumbent upon those who teach systems analysis and design to educate future systems analysts about open source development.

5. REFERENCES

Beck, Kent (1999) *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, ISBN 0-201-61641-6

Brooks, Frederick P. (1995) *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition*. Addison Wesley, ISBN 0-201-83595-9.

Dafermos, George (2001) "Management & Virtual Decentralised Networks: The Linux Project". Masters thesis, Durham Business School.

DeMarco and Lister (1987) *Peopleware: Productive Projects and Teams.* Dorset House, 1987, ISBN 0-932633-05-6.

Free Software Foundation (2005) "The Free Software Definition." Web document: www.gnu.org/philosophy/free-sw.html.

Gates, William (1976) "An Open Letter to Hobbyists." *MITS Computer Notes,* February, 1976. Currently available on the Worldwide Web at *www.blinkenlights.com/classiccmp/*

gateswhine.html.

Glass, Robert L. (2003) "A Sociopolitical Look at Open Source." *Communications of the ACM*, 46(11), November, 2003, pp. 21-23.

Mockus, Audris; Roy T. Fielding; and James Herbsleb (2000) "A Case Study of Open Source Software Development: the Apache server." *Proceedings of the 22nd International Conference on Software Engineering*, June 2000, Limerick, Ireland.

Mitsova, Helena and Markus Neteler (2004) "GRASS as Open Source Free Software GIS: Accomplishments and Perspectives." *Transactions in GIS*, 2004, 8(2), pp. 145-154.

Morton, Andrew (2004) "Open Source Software Development and the Software-using Business World." Transcript of a speech given at SDForum, November 16, 2004. Obtainable at www.groklaw.net/article.php?story=20041122035814276&query=Software-using+business+world

Open Source Initiative (2002) The Open Source Definition. WWW document, http://www.opensource.org/docs/definition.php

Raymond, Eric (1997) The Cathedral and the Bazaar. WWW document, www.catb.org/~esr/writings/cathedralbazaar/cathedral-bazaar

Raymond, Eric (1999) *The Magic Cauldron*. WWW document, www.catb.org/~esr/writings/cathedral-bazaar/magic-cauldron