School of Computer Science & Engineering
Faculty Publications

School of Computer Science and Engineering

2003

Ethical Issues in Open Source Software

Frances Grodzinsky
Sacred Heart University

Keith W. Miller
University of Illinois at Springfield

Marty J. Wolf
Bemidji State University

Follow this and additional works at: https://digitalcommons.sacredheart.edu/computersci_fac

Part of the Business Law, Public Responsibility, and Ethics Commons, and the Computer Sciences
Commons

Ethical Issues in Open Source Software

F S Grodzinsky
Sacred Heart University, Fairfield, CT, USA
Email: grodzinskyf@sacredheart.edu

K Miller
University of Illinois Springfield, Springfield, Illinois. USA
Email: Miller.keith@uis.edu

M J Wolf
Bemidji State University, Bemidji, MN, USA
Email: mjwolf@bemidjistate.edu

ABSTRACT

In this essay we argue that the current social and ethical structure in the Open Source Software (OSS) Community stem from its roots in academia. The individual developers experience a level of autonomy similar to that of a faculty member. Furthermore, we assert that the Open Source Software Community's social structure demands benevolent leadership. We argue that it is difficult to pass off low quality open source software as high quality software and that the Open Source development model offers strong accountability. Finally, we argue that Open Source Software introduces ethical challenges for universities and the software development community.

COVERAGE

1. INTRODUCTION

Open Source Software (OSS) and the emergence of an entire Open Source Movement have practical, political, economic and ethical ramifications for software development and software use. In this article we examine ethical issues that have been raised by open source software and its challenge to commercial software models. First we will trace the history and impetus of the open source movement, examining its forerunners, including UNIX, Stallman's Free Software Foundation, and Linux. Next we will define Open Source and examine its development model. Lastly, we present the ethical issues raised by OSS and attempt to demonstrate

how human values are interwoven with the economic and technical choices that OSS affords.

2. A BRIEF HISTORY OF SOFTWARE DEVELOPMENT

The field of software development, like many technological fields, has its roots intertwined with academia. Academia has a long-standing tradition of sharing ideas and results, as long as appropriate credit is given to the originators. This tradition is most easily observed in the rich collection of scholarly journals used by the research community. These journals provide a forum

KEYWORDS

Open Source Software

Linux

Software development models

GNU, FSF

for the dissemination of ideas and results. In addition, they provide a vehicle for researchers to advance new ideas and enhance the quality of existing theories. These traditions are deeply engrained in the Computer Science and Software Engineering academic communities.

The field of software development also shares a close connection with industry. The first ties occurred with hardware developers. IBM, for example, in conjunction with researchers at Harvard University, built its first computer in 1944. Over the next 15 years, IBM would bundle hardware, software and services in a single package, rarely distinguishing among the various components. Then in 1969 IBM adopted a new marketing policy in which software and services were marketed separately from hardware. Therefore, in order to have any opportunity for profitability for the programming division, the source code would need to be kept confidential. This separation became complete in 1981 when IBM partnered with Intel and Microsoft to develop the personal computer (The History of IBM). Hewlett Packard (HP) was another company that first marketed hardware and then later software. In 1966, HP's first computer was developed to control a variety of laboratory instruments. In 1969, HP marketed its first time-sharing operating system (About HP, 2003).

Perhaps the most interesting connection between software development, industry and academia surrounds the history of Unix. Unix was first developed in the early 1970's at AT&T Bell Labs in New Jersey, USA, largely due to the efforts of K. Thompson, D. Ritchie, M. D. McIlroy, J. F.Ossanna (Ritchie, 1996). One of their design goals was to develop a portable operating system. Thompson demonstrated the portability of Unix by mailing magnetic tapes containing Unix source code and utilities to friends (Moffitt,2002). At the same time, AT&T gave away source code and licenses to uni-

versities for Unix Level 6. AT&T even published two books that contained the source code along with commentary.

The Computer Science Department at the University of California, Berkeley, was one such licensee that used the source code extensively for research projects. Eventually, they enhanced the Unix source code to include many new features and were freely distributing their enhanced versions. People throughout the world added many additional features, some of which are used to run the Internet today. Throughout the 1980's there was confusion as to just what 'Unix' meant. There were lawsuits and counter lawsuits over which parts of Unix software source code could be freely given away and which parts required a royalty even to use in binary form. As we will see later, these issues largely went away in the early 1990's when Linus Torvalds introduced Linux.

While these developments were taking place in industry, the hacker culture was developing at many of the major research laboratories across the United States. The hacker culture involved people who loved to program and enjoyed being clever about it. By the early 1970's this culture was engrained at the Massachusetts Institute of Technology and had a significant impact on Richard Stallman's attitudes about software development. He says, "Whenever people from another university or a company wanted to port and use a program, we gladly let them. If you saw someone using an unfamiliar and interesting program, you could always ask to see the source code, so that you could read it, change it, or cannibalize parts of it to make a new program" (Stallman, 1999). The idea that source code could be freely exchanged, changed and used appealed to him as an efficacious software development technique. However, that system of development began to break down in the late 1970's and early 1980's as the changes noted above were taking place in industry. A further impact was the fact that industry was hiring many of the best software developers and programmers from the computing labs, and those individuals were taking the software they developed with them. To Stallman, succumbing to this industrial model of software development was tantamount "to making the world a worse place" (Stallman, 1999). In response to commercialization within the software development industry, Stallman began the

194

Perhaps the most interesting connection between software development, industry and academia surrounds the history of Unix.

GNU project in 1984 with a goal of "creating a new software sharing community" (Stallman, 1999). The GNU project goal was to develop an entire Unix-like operating system, complete with all the utilities, in order to build a community of developers dedicated to writing free software. All of the source code would be freely available for modification and use by anyone who was willing to make further changes.

The Free Software Foundation (FSF) was started in 1985 largely to support this objective (The GNU Project, 2002). By the early 1990's, the GNU project had succeeded in many ways. Useful software development and environment tools, plus the General Public License (GPL), had been developed under the auspices of the Free Software Foundation. However, GNU still lacked the core of an operating system – the kernel. However, when Linus Torvalds developed and released the Linux kernel under the GNU GPL the picture was completed. With the Linux kernel and the tools developed by GNU, the world now had a complete, functional operating system with all of the source code freely available for inspection, modification and improvement.

As the Linux kernel developed and matured, people began to take note of the software development methodology used to create it. In 1998, those who advocated the software development process that is afforded by shared source code started a movement called the Open Source Initiative (OSI). The Open Source Initiative shares many of the same goals as the Free Software Foundation (Open source initiative, 2002). However its focus is grounded in the development methodology that arises when source code is open and free to all who want it. On the other hand, the FSF is a proponent of a philosophy that puts the notion of free software first. According to the FSF there are four freedoms that are essential for free software:

1. Freedom to run the program, for any purpose.
2. Freedom to study how the program works, and adapt it to your needs.
3. Freedom to redistribute copies so you can help your neighbor.
4. Freedom to improve the program, and release your improvements to the public, so that the whole community benefits.

Requiring that all derivative works of GPL software also be licensed under the GPL, if and when they become published, propagates these freedoms. This protection is known as copyleft and prevents source code from being swallowed up in a commercial venture (See Appendix B).

The OSI is less focused on philosophical tenets emphasized by Stallman and more focused on promoting open source as a development methodology. This promotion is evident in the emergence of other important software cultures including the Perl culture under Larry Wall, John Osterhout's Tcl and Guido van Rossum's Python languages. "All three of these communities expressed their ideological independence by devising their own, non-GPL licensing schemes" (Raymond, 2000).

3. ETHICAL ISSUES IN OSS

What motivates a developer to write OSS? In the previous section we have alluded to the philosophy of Richard Stallman and members of the Open Source Initiative. Linus Torvalds believes that good software development starts by the scratching of a personal itch: the curiosity and desire to see if something can be done. In *The Cathedral and the Bazaar*, Eric Raymond depicts the OSS developer as one who has found the golden mean between underutilization and over-utilization caused by ill-formulated project goals. Raymond's "happy programmer," conforms to the Aristotelian model of one, who while enjoying the freedom to experiment and excel, brings forth imaginative and creative software (Raymond, 2001). Which, if any of these depictions still holds true? We will begin by examining the Open Source Software Development Community including the Open Source Definition as a social contract. Then we will explore some of the ethical issues that might explain the motivations of the OSS movement: autonomy, quality, and accountability. Finally we will close this section with analysis of whether Open Source can be considered a public good.

3.1 The Open Source Software Development Community

The Open Source Software development

195

community emerged out of a parting of the ways with Richard Stallman and the Free Software Foundation. While the details of the contentious debate are beyond the scope of this paper, we will try to articulate some of the major points. Stallman's vision was one of a community of programmers who were doing something for the good of humankind. Although he acknowledges that open source and free software belong to the same category of software, Stallman maintains that there was an ideological shift within those advocating for open source. Stallman asserts that while the GNU Project holds onto the concept of freedom, the OSS community tries to appeal to businesses. Because business values profit above all, he argues that values of community and freedom are lost in the OSS development model. Citing examples of proprietary software that work with Linux, Stallman wonders if OSS developers will shun or support them (Stallman, 1999).

Open Source advocates argue that OSS is primarily a development methodology grounded in the philosophy of making source code open and free to all who want it. Developers self-select according to their interest in a project. Users and developers co-exist in a community where software grows and expands based on personal needs. These enhancements make the project more globally desirable as it fits more and more requirements. Linus Torvalds, the epitome of the open source developer says:

- Release early and often
- Delegate everything you can
- Be open (Raymond, 2001, p.309).

It has been argued that with open source software "you get what you pay for."

It usually takes one interested developer to write a piece of core code to get a project going. Torvalds believes that it is the responsibility of the originator of the project to listen to the users who will find the bugs and those who will fix them. In this type of scenario, "users are rewarded by the daily improvement in the work" (Raymond, 2001, p. 315).

Creators of OSS typically use the software themselves as it is being developed; therefore, the users are involved in the software development from the beginning. When software is created solely for commercial gain, there is always a danger that the customer is treated merely as a means to a financial end: financial enrichment of the software developer. Open source software removes that temptation.

Empirical evidence suggests that developers are less concerned with the ideological split between the FSF and the OSS, than with the idea of making free software available to all. Eric Raymond explains, "A side effect of the rapid growth of Linux was the induction of a large number of new hackers for which Linux was their primary loyalty and the FSF's agenda primarily of historical interest" (Raymond, 2000). He maintains that it was the Netscape announcement in February 1998 that it would distribute Navigator 5.0 in source that changed the environment dramatically. The idea that 'free software' could be exploited within the commercial community excited the hacker culture and caused the parting of the ways between free software and open source advocates.

As with all software, quality is a major issue when evaluating Open Source Software. It has been argued that with open source software "you get what you pay for." If a developer is not paid for creating and maintaining a piece of software, then he/she may not feel the same obligation to do a quality job. Linus Torvalds argues that programmers tend to find projects that interest them, and if the programmer loses interest in a piece of code, there is usually a co-developer out there willing and able to continue on the project. The reason that Linux is so successful is that it interests people to develop a complete, open operating system. While it is true that there might be less success in developing a less glamorous project, developers would usually not undertake it, rather than do a shoddy job. Their reputations among their peers are at stake. In the hacker community, "one's work is one's statement...and there's a strong ethos that quality should (indeed must) be left to speak for itself...Boasting or self-importance is suppressed because it behaves like noise tending to corrupt the vital signals from experiments in creative and cooperative behavior". (Raymond, 2000).

Thus within the community of those working on the project, individual interest and work contribute to a larger goal: a working project for all. In addition, because it is open source, others may take advantage of the software once it is available on line. Floridi and Saunders in their paper entitled *Internet Ethics: the Constructionist Values of Homo Poieticus* maintain that a collaborative project relies on an "unsuspected but evident interest, shared by a growing community" and the coordination of efforts to produce a global product based on "local specific components." They name this phenomena 'distributed constructionism.' They maintain that the Internet facilitates communication amongst users irrespective of distance and creates an environment that encourages and facilitates production of OSS like Linux (Floridi *et al.*, 2003).

3.2   The Open Source Definition: A social contract

The formalization of the OSS community came about with the development of the Open Source Definition and the OSI. In 1997, Bruce Perens published a set of guidelines to articulate the developers' commitment to open source software and its users (Perens, 2002). The Debian Free Software Guidelines were incorporated into and became the basis of the Open Source Definition (See Appendix A). The OSI published licenses that met the Open Source Definition and declared that software distributed under any of these licenses would be 'OSI Certified.' The OSI offers copies of these licenses for anyone to use and modify for their own business model. The Mozilla Public License has been the one most often used since 1998. Most of these licenses give permission for the software to be used. Some ask that a copyright and year be included when redistributing the software. Most of them present the software 'as is' and indemnify themselves against liability.

The social contract articulated in the guidelines is fairly clear about what the OSS is offering to others. But what do OSS developers expect in return? What motivates developers to contribute to an open source project? Is it altruism, i.e., do they consider it a 'pro bono' project that contributes to the public good? Is it a reaction against corporate greed? Does it make them feel part of a select community with special talents? Clearly all of these play a part in OSS developer motivation to abide by this contract. Beyond that, however, there is also a sense that developers see their involvement as "enlightened self interest" (Kollock, 1999). Their contributions lead to software that they want and often need. OSS is an alternative for users; developers are themselves users of computing, almost always heavy users. Developers and users participate voluntarily in a development environment that emphasizes cooperation and mutual support under the OSI guidelines. They have found the environment satisfying enough to make OSS a major challenger to commercial software.

3.3   Autonomy

One perceived attraction for OSS developers is the autonomy of the developer. While developers who embrace OSS do gain a measure of autonomy not available to developers working on commercial software, the claim for *complete* autonomy doesn't appear to be valid. OSS developers work as volunteers, and can join or quit an effort strictly on their own initiative. These volunteers are not coerced into participation and willingly contribute. Therefore, one might assume that the OSS developer can be depicted as a libertarian ideal, unshackled by corporate controls. However, there are several types of control in OSS, even though no single developer is in charge of an OSS project. As an OSS developer, the developer cannot be sure that his/her contribution will be accepted into the canonical version that is continuously evolving. This contribution may be embraced or rejected in the short term, and if accepted may be changed or replaced later. The developer is free to contribute or not, but any single developer cannot claim ultimate control over the use of his/her contribution. In *Homesteading the Noosphere,* Eric Raymond states, "the open-source culture has an elaborate but largely unadmitted set of ownership customs. These customs regulate who can modify software, the circumstances under which it can be modified, and (especially) who has the right to redistribute modified versions back to the community"(Raymond, 2000).

197

Open source software has the seemingly useful feature that at any point, any one with appropriate technical skills can modify the code and take the project in a direction that diverges from the direction others are taking it (called 'code forking'). Thus, one of the perceived benefits of a piece of

> **Open source project leaders and developers must show a great willingness to take in new ideas**

198

open source software is that it has the opportunity to evolve rapidly into competing programs where, presumably, Darwin's theories of evolution can take over: the best piece of software for the current environment will survive. If code forking is prevalent, we might expect to see many innovations occur in open source software development. However, Raymond gives two very pragmatic reasons for the low incidence of code forking in many successful OSS projects. The first major reason for projects to persist is a fear of diluting the developer community for the project -- both child projects have fewer developers, thus weakening the entire project, especially relative to the parent project. Secondly, shortly after a code fork the child projects cannot exchange code. In addition, Raymond adds that "there is strong social pressure against forking projects" in the open source community. According to Raymond the open source community is best viewed as a gift culture where one's social status is determined by what one gives away. In addition to the practical concerns, forking a project cuts right to the core of the culture-it damages someone's reputation.

Thus, given both the pragmatic and cultural pressure to avoid code forking, the developers of an open source project must take special care to avoid the symptoms of groupthink. A newcomer to open source development has very little in terms of reputation to bring to the table when he/she proposes a new piece of code or a new tack on development for a project. Project leaders, who are less open to new ideas and

ways of doing things, may miss the innovation of the newcomer's idea. Not only will the project lose the good idea, but it also faces the potential of losing a good developer. Thus, open source project leaders and developers must show a great willingness to take in new ideas, evaluate them thoughtfully, and respond constructively in order to nurture both the idea and the developer of the idea.

Project leaders must exercise similar abilities when a subgroup comes with an idea that is controversial. Care must be taken that the larger group does not ride roughshod over the smaller group's idea. Again, in addition to losing out on a good idea and potentially driving people away from the project, doing so will discourage future innovators from taking their ideas forward. Note that the proprietary software development model is not subject to this argument. The innovative developer who meets resistant project leaders or management is typically free to leave the organization, and he/she regularly does. In fact there are social norms that actually encourage this type of behavior; we call these people entrepreneurs.

So it appears that the autonomy experienced by an open source developer is much like the autonomy experienced by a university faculty member: freedom to choose which projects to work on. Thus, an open source developer has increased autonomy when compared to a corporate developer. Whereas, the corporate developer might find a supportive social structure to take a project in a new direction, the social structure in the Open Source community can work to suppress this type of entrepreneurial endeavor.

3.4   Software quality

Quality software, in the traditional sense, is software that meets requirement specifications, is well tested, well documented and maintainable (Schach, 2002). Advocates of OSS claim that its developers/users are motivated to do quality work because not only are they developing software for their own use, but their reputations among their peers also are at stake. Critics of OSS claim that volunteers will not do professional quality work if there is no monetary compensation. They also claim that documen-

tation and maintenance are non-existent. While it is true that documentation and maintenance are concerns, OSS advocates assert that OSS meets users' requirements, is tested by its developers and is constantly being upgraded. Documentation evolves as more and more users become interested in the software and use it. For example, books on Linux can be found everywhere.

The question of whether OSS is of higher or lower quality than comparable commercial software is essentially an empirical rather than philosophical question. The answer to this question is not readily available, but we can cite some preliminary anecdotal evidence on this issue. The Apache web server is OSS that competes with commercial web servers. The web server market is a potentially lucrative one, and we expect commercial software developers to compete for that market with high quality software. Yet despite commercial alternatives, according to third party observers (Netcraft, 2002) the OSS Apache server is by far the most used web server.

According to an August, 2002 survey, 63% of web servers on the Internet are Apache. At least in this market segment, it appears that OSS is sufficiently high quality for most users. Of course, Apache is free and other servers aren't; the cost motivation might explain some of Apache's popularity. But if the Apache server were of significantly lower quality than commercial alternatives, then it would be surprising to see its widespread use. This raises the question of whether market-dominance and popularity should be a benchmark for software quality. Does the fact that Microsoft Windows runs on some 90% of home computers assure us of its quality? We would argue that popularity and quality might be linked if it can be shown that there is a level of expertise about software quality in the people making the choices. System administrators have more expertise than an average user of a home computer system. Therefore, when a majority of these professionals choose an OSS alternative, it deserves notice.

The Apache example illustrates an important distinction among OSS users. Initially, first adopters of OSS are its developers and as the code becomes more known, OSS gains users who were not involved in the development. These users adopt the OSS for many reasons, but some of these new users (particularly non-programmers), appreciate the product, though they may not understand or care about the process that developed it. All users of OSS gain if the software delivers needed functionality.

If an OSS project pleases its developers, but does not gather a following outside the

> **The OSS model has different kinds of successes, and fewer outright failures.**

developing community, that may be fine with the developers; if a commercial project only pleases its developers, it is a financial failure. The OSS model has different kinds of successes, and fewer outright failures. The rewards for developers in an OSS project are likely to be less tangible than rewards for a successful commercial product, but that does not make the rewards less real. The public has potential gains in the OSS movement that do not require large investments by the public.

Another distinction between OSS projects and commercial projects is the lack of a release date. While open source developers anticipate frequent releases, there are no release deadlines. The announcements of a release day by a commercial vendor impose pressure on developers to cut corners, thus increasing the possibility of errors in the software. Furthermore, such a deadline has a tendency to impose on the autonomy of the developer.

Finally we note that both open source and proprietary developers share the same professional ethical responsibility to develop solid, well-tested code. The social pressure in the open source community to avoid code forking provides incentives for project leaders to ensure that the code is the best it can be. On the other hand, when an open source developer believes there is too much risk associated with a particular piece of code, he/she can rewrite it and release it. While there is a reputation risk in doing so, there is the opportunity to publicly demonstrate that the forked product is superior. In a proprietary model, however, a developer's main avenue of

199

recourse is to 'blow the whistle' on his/her manager or employer. To do so entails grave personal risk to one's livelihood, professional standing, lifestyle and family. Worse yet, the developer will likely not have the opportunity to demonstrate the wisdom of his/her ways.

3.5 Open Source and Accountability

In her article entitled *Computing and Accountability,* Helen Nissenbaum cites four barriers to accountability:

1. The problem of many hands,
2. bugs,
3. computer as scapegoat and
4. ownership without liability.

She asserts that these barriers can lead to "harm and risks for which no one is answerable and about which nothing is done" (Nissenbaum, 1994). We will examine how OSS may have addressed barriers 1 and 2. Number 3 is a general issue and number 4 does not apply because there is not software ownership per se in open source. The Open Source Definition #1 addresses her fourth point (See Appendix A).

"Where a mishap is the work of 'many hands', it can be difficult to identify who is accountable because the locus of decision making is frequently different from the mishap's most direct causal antecedent;

Accountability is not lost in the group, but is instead taken up by the entire group

---

that is, cause and intent do not converge" (Johnson, 1995). In open source, however, if a developer were to write irresponsible code, others contributing to the open source software would be unlikely to accept it. So, in this case, there is built-in individual accountability. If a developer were part of a large company, where all programming parts contribute to a large commercial venture, it then would fall on both the company and the individual to accept responsibility for the problematic software product.

Often this is not done. So the many hands problem referred to by Nissenbaum in Computing and Accountability can be reduced in OSS because parts of code can be ascribed to various developers, and their peers hold them accountable for their contributions.

Nissenbaum argues that accepting bugs as a software fact of life has issues regarding accountability (Nissenbaum, 1994). The open source approach to software development treats the bug problem with a group effort to detect and fix problems. Torvalds states, "given enough eyeballs, all bugs are shallow" (Raymond, 2001, p. 315). The person that finds a bug in OSS may not be the person to fix it. Since many adept developers examine OSS code, bugs are found and corrected more quickly than in a development effort where only a few developers see the code. In this group effort, accountability is not lost in the group, but is instead taken up by the entire group. The question of whether this group accountability is as effective as individual responsibility is, again, empirical. The examples of Apache and Linux (Webcab solutions, 2003) offer at least anecdotal evidence that some OSS demonstrates high reliability.

Don Gotterbarn is also concerned about issues of professional accountability in OSS (Wolf *et al*, 2002). In addition to worries about sufficient care in programming and maintaining OSS, Gotterbarn points out that an OSS licensing agreement forces the authors of the software to relinquish control of the software. If someone puts OSS to a morally objectionable use, then the developers have no right to withdraw the software from that use.

Gotterbarn's objection has some theoretical interest, for the OSS licensing agreements clearly state that no one who follows the OSS rules can be blocked from using the software. But if we accept the idea that software developers have a moral duty to police the use of the software they distribute, especially when the software is utility software, we fall into a practical and theoretical thicket. How is a vendor to know the eventual use of software, especially when the software is utility software (such as an operating system or a graphics package)? Are software developers empowered to judge the ethics of each customer or perspective customer? These responsibilities are overreaching ethically, and far too

ambitious in a practical sense.

Furthermore, the relinquishment of control argument has practical significance only if existing competing software models include effective control over the use of software. (That is, should OSS be held to a higher standard than commercial software in relation to ethical responsibility for downstream use?) We are unaware of any action by existing commercial software vendors to police the uses to which their software is put. Commercial software vendors are certainly concerned that people who use their software have paid for it. Once paid for, vendors concerned about ethical use do not police commercial software.

3.6 Is Open Source a Public Good?

The claim that OSS is a revolutionary idea, a departure from previous models of intellectual property, is worth examining. Although clearly distinct from a commercial model of software development, OSS can be seen as a continuation of previously accepted traditions in academics in general, and in mathematics in particular.

Academia has long had the tradition of sharing ideas without direct payments. Scholarly journals do not pay authors (and in fact may charge them for pages printed). Law has not protected mathematical formulae and formal descriptions of natural laws. Copyright covers the expression of ideas, but not the ideas themselves; patent has (at least traditionally) protected the practical application of ideas, but not the physical laws underlying the ideas. So, if software is viewed as an extended mathematical object, akin to a theorem, then OSS could be a natural extension of the long tradition of free ideas in mathematics. Does that make it a public good?

Peter Kollack, a sociologist at the University of California at Los Angeles, examines the idea of public goods on line in his paper entitled *The Economies of Online Cooperation: Gifts and Public Goods in Cyberspace*. He defines public goods as those things that are non-excludable and indivisible. Because the Open Source Definition prohibits discrimination against persons or groups or against fields of endeavor (see Appendix A), it supports the definition of a public good being non-excludable. Public goods in cyberspace can benefit the users of cyberspace irrespective of whether they have contributed to these goods or whether these goods have come from groups or individuals. The fact that one person using OSS does not affect its availability to the whole supports Kollack's idea of indivisibility. He maintains that "[a]ny piece of information posted to an online community becomes a public good because the network makes it available to the group as a whole and because one person's 'consumption' of the information does not diminish another person's use of it" (Kollock,1999). If a user downloads a copy of Linux, for example, it does not diminish its availability for other users. So by this definition, we concur that OSS is a public good.

However, is there an active interest among developers to create a public good? Are OSS developers actually motivated to do good by contributing software to the public, and by maintaining it in a group effort? Some developers argue that they can customize OSS, and if others find the customizations useful, then they have provided a public good. However, there could be another possible motivation for OSS. It might be a philosophical or instinctive animus towards existing commercial software developers. Bertrand Meyer recites with dismay the many negative statements by OSS advocates about commercial software development and developers (Meyer, 2000). Some see 'Microsoft bashing' as a central theme of the OSS movement. Since

> However, is there an active interest among developers to create a public good?

most OSS competes directly with Microsoft products, some friction between OSS advocates and the largest commercial software corporation seems inevitable. But if OSS development is motivated primarily by its opposition to commercial software producers, then its ethical underpinnings are less benign than if OSS is motivated primarily by an altruistic desire to help

201

computer users. Since the OSS movement is, by design, decentralized and evolving, it seems impossible to gauge with any precision the motivations of all its members. But the often-repeated disdain for commercial business practices seems more in tune with the hacker culture than with a culture of altruism. So, we would argue that for the most part, the altruism involved in the creation of a public good in the case of OSS is more of a by-product of developers who are interested in creating tools that are of use for themselves. Customization and expansion of Linux, for example, came from developers who wanted applications for their own use and then shared their code.

Nowhere can OSS be considered more of a public good than in the academic community. Computer Science departments are expected to be on the cutting edge of technology in their curricular offerings.

The price of commercial software, even with educational discounts, often straps a department's budget. Academic institutions have strong financial motivations to adopt open source software. GNU compilers, for example, have largely replaced proprietary versions that cost the university software fees as well as licensing fees. Linux is appearing as the operating system of choice often replacing Solaris. As more and more applications run on Linux, universities will have less incentive to buy from vendors who offer a UNIX platform. They will buy cheaper hardware and run Linux. One caveat to this scenario is the availability of staff that can support the Linux platform and the availability of documentation for OSS.

Using OSS at a university raises interesting ethical questions. One could argue that a university should expose its students to multiple perspectives so students develop skills to make judgments about the world around them. A university that exposes its students to a single point of view fails to help a student develop these skills. Thus, a university should provide a learning environment where future computer science professionals are exposed to both proprietary and open source software, and be given experiences to develop software evaluative skills. Part of this evaluation would come from using the software and evaluating the effectiveness from a user's perspective. However, an important part of the evaluative process, at least for computer

science students, involves accessing the source code. Open source software makes looking at the source code easy. While it is true that some proprietary software vendors are willing to share their source code with universities for educational purposes, others are not. It is precisely when the software vendor is unwilling to share source code with university faculty, or makes it onerous on the university, that the university is faced with an ethical dilemma: How does it respond to proprietary software vendors that interfere with its duty to educate its students? OSS provides a partial solution to that problem. These questions are further complicated by the relationship that software companies often share with many universities. It is not uncommon for a software company to make generous contributions to a computer science department for access to the department's graduates, or to sponsor a faculty member's research. Faculty at these institutions must take care not to let the largess interfere with their ethical responsibilities to their students.

If a university is part of the open source community, we might expect them to be a contributor as well as a user of OSS. For many places of higher education, especially research institutions, this is not an issue as university faculty and students develop much open source software. But those institutions, whose faculty and students are not making such contributions, are faced with making the choice of contributing in some other way.

One approach might be a cash donation to an appropriate foundation that supports open source development for the value the software brings to the educational environment. This is not always easy to do. Anecdotally, a federal employee who wanted to do this reported two problems:

1. Accountants in her agency balked at making a contribution; they thought it might be illegal.
2. She tried but could not get additional clarifying information from the OSS foundation. Contact people listed at the foundation did not return her emails and phone calls. She ended up not making a donation and feeling bad about it.

It may be argued that merely exposing students to open source software may fulfill the university's ethical obligation to support OSS

since doing so meets the OSS community's goal of building the OSS user community.

Finally, we explore an issue that is becoming part of the mission of many institutions of higher education: service learning. The choice between open source software and proprietary software plays into service learning as well. Consider a scenario where a software engineering class is to produce a piece of software for a local charity. The choice between open source alternatives and proprietary alternatives is not to be taken lightly. Seemingly, open source software makes good sense for both the students and the charitable organization. The cost is low and, presumably, the quality is sufficient. Yet there are long-term costs that are faced by the charity (as well as any business making such a choice). How expensive will it be to maintain the software? Is there enough open source expertise available to maintain it? And, finally, what documentation and user training can be expected if OSS is the software of choice. An extension of the service model might offer some on-going support to these charities.

4. CONCLUSION

OSS is no longer an academic curiosity. We have demonstrated that certain OSS products are making a significant niche for themselves in computing environments. Both Apache and Linux are increasing in popularity. The OSS model is distinct from commercial software development from several viewpoints: as a software engineering process, as an economic plan, and as a marketing strategy. In both models, however, developers have certain obligations and responsibilities to their users. In our analysis we have argued that open source software's successes may be due in part to the sheer number of people who get involved, and to the users who are engaged from the start of development.

We have found that the authors of OSS have complex motivations, some laudatory, and others less so. OSS has produced some successes, and the public has benefited from these. There are questions about reliability and professionalism, but evidence against the quality of OSS is not, as yet, convincing to us. It does not appear likely that OSS will displace commercial software

in the foreseeable future, and we have not uncovered any ethical imperative that it should. Yet, OSS has distinct economic

---

> **All software developers have ethical obligations for quality and openness.**

---

advantages for many especially in the academic arena. It can help bridge the digital divide and can involve growing numbers of people in computing, both as developers and users. Developers of OSS strive to be the best they can to contribute to the sustainable whole and thus secure their reputation ethically among their peers.

OSS and commercial software can coexist, each giving the public the goods it desires. Both advocates and critics of OSS have an ethical obligation to respect each other and to avoid inaccurate and mean-spirited accusations. All software developers have ethical obligations for quality and openness (Software engineering code of ethics, 1999). OSS is a novel development of traditional ideas of sharing academic intellectual property, but OSS exists in a world dominated by commercial enterprise. As such, OSS challenges the status quo in a way that can be a constructive check on excesses of traditional free enterprise systems. In a time when many for-profit corporations have disappointed the public with their lack of ethical behavior, OSS has the potential to be a positive ethical force in the world of computing. Hackers who get involved in OSS development can contribute to the sustainable whole and, thus ethically secure their reputation among their peers. This is a way to publicly excel at hacking without illegal and unethical harm to others.

APPENDIX A

Open Source Definition

Open source doesn't just mean access to the source code. The distribution terms of open-source software must comply with the following criteria:

A.1   Free Redistribution

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

A.2   Source Code

The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost-preferably, downloading via the Internet without charge. The source code must be the preferred form in which a developer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

A.3   Derived Works

The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

A.4   Integrity of The Author's Source Code

The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

A.5   No Discrimination Against Persons or Groups

The license must not discriminate against any person or group of persons.

A.6   No Discrimination Against Fields of Endeavor

The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

A.7   Distribution of License

The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

A.8   License Must Not Be Specific to a Product

The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

A.9   The License Must Not Restrict Other Software

The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.: http://www.opensource.org/docs/definition_plain.html

APPENDIX B

"To copyleft a program, we first state that it is copyrighted; then we add distribution terms, which are a legal instrument that gives everyone the rights to use, modify, and redistribute the program's code or any program derived from it but only if the distribution terms are unchanged. Thus, the code and the freedoms become legally inseparable."(www.FSF.org)

204

REFERENCES

About HP: History and Facts. www.hp.com/hpinfo/abouthp/histnfacts/. Accessed 2003.

Barr, J. Live and let license: A primer on software licensing in the Open Source context, May 23, 2001.

Computers, Ethics and Social Values. Johnson, D.J. and Nissenbaum H., eds. Prentice Hall: New Jersey, 1995.

Floridi, L. and Sanders, J.W. Internet Ethics: the Constructionist Values of Homo Poieticus. In *The Impact of the Internet on Our Moral Lives*, Cavalier, R., ed. New York: SUNY, 2003.

Kollock, P. The Economies of Online Cooperation: Gifts and Public Goods in Cyberspace. In *Communities in Cyberspace*, Smith, M. and Kollock, P., eds. London: Routledge, 1999.

Meyer, B. The Ethics of Free Software. *Software Developers Online.* www.sdmagazine.com/documents/s=746/sdm0003d/0003d.htm, login required, March 2000.

Miller, R. 90% Windows, 5% Mac, 5% Linux? Not true! *The Register.* www.theregister.co.uk/content/4/19661.htm, June 13, 2001.

Moffitt, N. Nick Moffitt's $7 History of Unix. www.crackmonkey.org/unix.html. Accessed 2002.

Netcraft Web Server Survey. www.netcraft.com/survey, 2002.

Nissenbaum, H. Computing and Accountability. *Communications of the ACM,* 37, 1, January 1994.

Open Source Initiative (OSI). www.opensource.org, 2002.

Perens, B. Debian Social Contract. www.debian.org/social_contract.html, 2002.

Raymond, E.S. *Homesteading the Noosphere.* tuxedo.org/-esr/writings/cathedral-bazaar, 2000.

Raymond, E.S. "The Cathedral and the Bazaar", in *Readings in Cyberethics*, eds. Spinello and Tavani, Jones and Bartlett, 2001.

Ritchie, D.M. The Evolution of the Unix Time-sharing System. cm.bell-labs.com/cm/cs/who/dmr/hist.html, 1996.

Schach, Stephen, *Object Oriented and Classical Software Engineering* (fifth ed). McGraw Hill, 2002. p. 137.

Software Engineering Code of Ethics and Professional Practice(5.2). seeri.etsu.edu/TheSECode.htm, 1999.

Stallman, R.M. The GNU Operating System and the Free Software Movement. In Open Sources: Voices from the Open Source Revolution, Stone, M., Ockman, and DiBona, C., eds. , 1999.

The GNU Project and the Free Software Foundation (FSF). www.gnu.org and www.fsf.org, 2002.

The History of IBM. www-1.ibm.com/ibm/history/index.html Accessed spring 2003.

WebCab Solutions – Linux Reliability. Accessed 2003.

Wolf, M.J., K. Bowyer, D. Gotterbarn, and K. Miller. Open Source Software: Intellectual Challenges to the Status Quo, panel presentation at *2002 SIGCSE Technical Symposium, SIGCSE Bulletin*, 34(1), March 2002, pp. 317-318.

205

CORRESPONDING AUTHOR

Fran Grodzinsky
Sacred Heart University, Fairfield, CT, USA
Email: grodzinsky@sacredheart.edu