CHAPTER 10

Ethical Interest in Free and Open Source Software

FRANCES S. GRODZINSKY and MARTY J. WOLF

10.1 INTRODUCTION

Free Software (FS), a concept developed by Richard Stallman in the 1980s, has served as a foundation for important and related movements that have become possible because of the Internet. The most important of these has been the Open Source Software (OSS) movement. OSS, a concept rooted in software methodology and analyzed by Eric Raymond, broke from the FS ethos in 1998. This paper will compare FS and OSS, examining their histories, their philosophies, and development. It will also explore important issues that affect the ethical interests of all who use and are subject to the influences of software, regardless of whether that software is FS or OSS. We will argue that the distinction between FS and OSS is a philosophically and socially important distinction. To make this point we will review the history of FS and OSS with a particular emphasis on four main people: Richard Stallman, Linus Torvalds, Eric Raymond, and Bruce Perens. In addition, we will review the differences between GNU[1] General Public License (GPL) version 2 (v2) and the current draft of the GPL version 3 (v3), and the related controversy in the OSS community. The GPL is the primary mechanism used by the software community to establish and identify software as free software. In section 10.3, we will examine the motivation and economics of OSS developers. We will review issues of quality with respect to OSS, autonomy of OSS software developers, and their unusual professional responsibilities. The final important issue we address is consideration of OSS as a public good.

[1]GNU is a recursive acronym for GNU's not Unix.

The Handbook of Information and Computer Ethics, Edited by Kenneth Einar Himma and Herman T. Tavani

10.2 ON THE DISTINCTION BETWEEN FS AND OSS

10.2.1 The History of Free and Open Source Software

Free software stems from the close ties that early software developers had with academia. As the software industry began to mature, the bond with academia and its ideals of sharing research results weakened. After spending many years as an active participant in the hacker culture, Richard Stallman grew frustrated as more and more software was not free—not free in a financial sense, but free in a way that allowed for its inspection, running, and modification. Stallman took a stand and began the GNU project in 1984. The goal of the project was to establish a software development community dedicated to developing and promoting free software. He established the Free Software Foundation (FSF) to support his plan to create an operating system complete with all of the tools needed to edit, compile, and run software. This effort resulted in a large collection of free software. As part of this work, he codified his notion of free software in the GNU General Public License. Stallman was (and still is) vocal in articulating a moral argument for free software and developing free software as a viable alternative to nonfree software. In the early 1990s, Linus Torvalds was instrumental in further strengthening the viability of free software when he licensed his Linux operating system kernel under the GPL. When this kernel was bundled with GNU's software tools, Stallman's goal of a completely free operating system was achieved.

Although there was an active and productive worldwide community surrounding the GNU/Linux operating system, free software failed to gain much traction in the corporate setting. Eric S. Raymond was instrumental in demystifying many aspects of free software in his essay "The cathedral and the bazaar." This essay motivated Netscape to consider making their browser software free. However, business concerns took hold, and they were unwilling to make the move completely. After consultation with Raymond and others, Netscape released the source code for their browser as *Open Source Software*. It was at this time that Raymond and Bruce Perens founded the Open Source Initiative (OSI). They established a definition of open source software (The Open Source Definition, 2006), distinguishing it from free software. Whereas the two notions are closely related, free software is quite rigid in its definition. There are four basic freedoms, including the freedom to modify and redistribute the software, that cannot be impinged upon. Authors of OSS, however, can place certain restrictions on modifications to and the distribution of the modified software. The history of both free and open software is more fully developed in numerous places, including Grodzinsky et al. (2003). In the next sections we explore deeper distinctions between the free software and open-source software communities. These two communities, although deeply intertwined and closely related, have distinct goals that clearly manifest themselves in the discussion surrounding the release of a draft of the next version (version 3) of the GPL.

10.2.1.1 Free Software Richard Stallman first articulated the ideals of the Free Software movement in 1985 in *The GNU Manifesto* (Stallman, 1985). In it he

articulates his motivations for starting the GNU[2] project and lays the groundwork for the GNU General Public License. In particular, he notes that "[e]veryone will be permitted to modify and redistribute GNU, but no distributor will be allowed to restrict its further redistribution" (Stallman, 1985). It is this notion that served as the foundation for the definition of free software.[3] He argues that all computer users would benefit from the GNU project and the GPL because effort would not be wasted redeveloping software; everyone would be able to make changes to suit his/her own needs; educational institutions would be able to use the software to help students learn about software; and no one would be burdened with the responsibility of deciding who owns which piece of software and exactly what one is allowed to do with it.

Stallman's exact position on the ethics of free software is unclear. He articulates his responsibility as a software developer in the *Manifesto*: "the golden rule requires that if I like a program I must share it with other people who like it" (Stallman, 1985). In a later essay he extends that responsibility by arguing that "programmers have a duty to write free software" (Stallman, 1992). He also seems to articulate a view that selling software is morally wrong. "Software sellers want to divide the users and conquer them, making each user agree not to share with others" (Stallman, 1985). However, in this early paper it is unclear whether he is considering all software or just "infrastructure" software, for example, operating systems, networking software, software development tools, because many of his arguments focus solely on GNU. Later, though, he states his position more pointedly and goes even further, claiming that "proprietary software developers" who obstruct the use of that software by users "deserve a punishment rather than a reward" (Stallman, 1992).

In the *Manifesto*, he also deals with some of the early objections to free software. We mention those with substantial ethical importance here. The first objection centers on programmers being rewarded for their creativity. He makes a distinction between deserving a reward and asking for a reward. He states that "[i]f anything deserves a reward, it is social contribution," and that "[t]here is nothing wrong with wanting pay for work" (Stallman, 1985). However, he insists "the means [of charging for software] customary in the field of software today are based on destruction" (Stallman, 1985). He argues that by asking users to pay for software,[4] certain people will not be allowed to use the software, resulting in reduced benefit to humanity. "Extracting money from users of a program by restricting their use of it is destructive because the restrictions reduce the amount and the ways that the program can be used. This reduces the amount of wealth that humanity derives from the program" (Stallman, 1985). In a later essay he makes the assumption that "a user of software is no less important than an author or even the author's employer" (Stallman, 1992). He acknowledges that not everyone

[2]GNU is a piece of software designed to have the same functionality as Unix and be completely compatible with Unix.

[3]Note that *free* refers to freedom, not price.

[4]Early in this movement, Stallman lacked clarity regarding free software. He often merged the notions of "no cost" and "freedom." Similarly, he seems to confuse the notions of software with restrictive proprietary licenses and charging for software.

may agree with him on this point.[5] However, he argues that those who do are logically required to agree with his conclusions.

A second objection to free software that Stallman deals with in the *Manifesto* is that a programmer has a right to control the results of his/her creative endeavor. Stallman argues that by controlling one's software, one exerts "control over other people's lives; and it is usually used to make their lives more difficult" (Stallman, 1985). When Stallman talks about control here, he is talking about the fact that under proprietary software licenses, users of software are typically restricted from making copies for others and making modifications to the software to meet their own needs. A potential software user, though, needs to weigh the difficulties faced without the software against the difficulties faced when the software is used. Assuming the user purchases the software with full knowledge of the terms and conditions, the user has not been taken advantage of as Stallman suggests.

Stallman brings a social justice bent to this objection as well. "All intellectual property rights are just licenses granted by society because it was thought . . . that society as a whole would benefit by granting them" (Stallman, 1985). Stallman seems to be of the opinion that once you buy a piece of software, you should have rights to control it, much like you would when you purchase a book. He notes that the notion of copyright did not exist in ancient times; it was created in response to technological developments (i.e., the printing press) and was used to prevent businesses from exploiting authors. Society benefited from copyright because authors, knowing they could control the mass production of their works, had sufficient incentive to produce creative works. Stallman sees proprietary software developers taking advantage of the copyright system because the public is unaware of the trade-offs it made in establishing the system. He makes the case that the general population has not examined why it values intellectual property rights. "The idea of natural rights of authors was proposed and decisively rejected when the US Constitution was drawn up. That's why the Constitution only *permits* a system of copyright and does not *require* one; that's why it says that copyright must be temporary. It also states that the purpose of copyright is to promote progress—not to reward authors. Copyright does reward authors somewhat, and publishers more" (Stallman, 1994). His point is that society has not thought thoroughly about copyright for some time, and this issue is not being dealt with honestly by copyright holders. "At exactly the time when the public's interest is to keep part of the freedom to use it, the publishers are passing laws which make us give up more freedom. You see copyright was never intended to be an absolute monopoly on all the uses of a copyright work. It covered some uses and not others, but in recent times the publishers have been pushing to extend it further and further" (Stallman, 2001).

[5]There are those who clearly disagree with Stallman. Himma denies that user interests necessarily win out over creator interests. He argues that content creators invest the most precious resources of their lives, time and effort, in creating content, while the most important interests of users in such content is frequently, but not always, that they merely want the content. Although the fact that someone wants something is of moral significance, Himma argues that, from the standpoint of morality, the content creator's interest in her time and effort (and hence in the content she creates) wins out over mere desires of others (Himma, 2006, 2008).

Ultimately, Stallman sees the social value of an individual modifying and sharing a program as more valuable to society than the author's intellectual property rights.

By the time that version 2 of the GPL was introduced in 1991, Stallman was much clearer in his pursuit of the four freedoms that are essential for free software (although this definition did not appear until 1996):

(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.

In addition to the four freedoms, GPLv2 also introduced a notion called "copyleft." Copyleft is a play on the word "copyright," but, more importantly, it leverages copyright law to propagate the four software freedoms. In particular, it requires that derivative works also be licensed under the GPL. Thus, once a piece of software is made free by the GPL, it and all of its derivative works will always be free. Thus, the GPL is the main mechanism for establishing and propagating software freedom. Later, we will consider the viewpoint that copyleft is coercive because the legal weight of the copyright system is used to force others to propogate free software.

With the clear articulation of the four freedoms (and a mechanism to spread them), Stallman was in a position to argue more clearly for free software. In *Why Software Should Be Free*, he clearly explains the social cost of software having owners (Stallman, 1992). He is careful to separate out the act of creating software from the act of distributing software. He argues that once software is created, society is harmed in three ways when software is not distributed freely: software is used by fewer people, software users are unable to adapt or fix the software, and the software cannot be used to learn from to create new software. He uses a utilitarian argument to suggest that proprietary software is an unethical choice. The purchase of software is zero-sum—wealth is transferred between two entities. "But each time someone chooses to forego use of the program, this harms that person without benefiting anyone" (Stallman, 1992). He goes on to claim that the decision by some not to purchase software harms society because those people do not derive the benefit of that software. Stallman's point is subtle here. Distributing software, unlike distributing material goods, requires no new raw materials or packaging and the incremental distribution costs to allow widespread use are zero or very small.

Stallman also argues that the typical proprietary license damages social cohesion because it restricts one neighbor from helping another. Such a license demands that a person give up the right to copy the software in the event that a neighbor would benefit from its use. The damage comes because people "know that they must break the laws in order to be good neighbors" (Stallman, 1992). He thinks that the software copyright system reinforces the notion that we must not be concerned with advancing the public good. "[T]he greatest scarcity in the United States is not technical innovation, but rather the willingness to work together for the public good" (Stallman, 1992).

A related, weaker claim by Stallman is that proprietary programmers suffer harm in knowing that everyone, quite possibly even themselves, cannot use the software in the case that the owner is the author's employer. Stallman has two additional arguments regarding the social cost of keeping software proprietary. The first is a slippery slope argument that proprietary software begins to destroy the ethic of making contributions to society. This argument is suspect. It is no secret that Bill Gates has made enormous sums of money from proprietary software. The Bill and Melinda Gates Foundation is evidence that the Gates' ethic of making contributions to the greater good is still intact.

Stallman also argues that there is a social cost of frustration and lost capital because of proprietary software. Writing replacement software is frustrating for the programmer and more expensive than modifying and improving existing software (Stallman, 1992). This argument is weak, because there is social value in having two competing pieces of software that are functionally equivalent. It seems that the apparent robustness and security of GNU/Linux has prompted Microsoft to take robustness and security more seriously in the Windows operating system. Also, there are numerous software categories where there are competing FS packages. For example, both KDE and Gnome (both desktop software) have their ardent supporters. Each provides the same functionality (at least on a high level) and allows users and other developers to choose software that most appropriately meets their needs.

In a later essay, *Why Software Should Not Have Owners*, Stallman analyzes arguments for software ownership (Stallman, 1994). He notes that "[a]uthors often claim a special connection with programs they have written" (Stallman, 1994) and because of that special connection, in ethical analyses, software authors' positions should bear more weight. Proponents of this argument claim that this connection comes from extending rights associated with material objects to software. Stallman asserts that material objects are fundamentally different from software and that there is no evidence that software is deserving of the same protection. The fundamental difference stems from the scarcity of material objects relative to the (infinite) abundance of software. As mentioned earlier, it is easy and cheap to make copies of source code without depriving the holder of the source code access to the original copy. Again appealing to an act utilitarian analysis he notes, "[W]hether you run or change a program I wrote affects you directly and me only indirectly. Whether you give a copy to your friend affects you and your friend much more than it affects me" (Stallman, 1994). It might be argued that the last statement is not true. The holder of the original is deprived of the profit that would have been made through selling the software, but this is largely an economic argument.

10.2.1.2 Open Source Software The free software community grew substantially after the introduction of the Internet and Linus Torvalds' contribution of Linux as free software. It quietly made gains, without garnering widespread attention, until 1998, when Eric Raymond and Bruce Perens teamed to create the Open Source Initiative. In 1997, Raymond gave the first thorough analysis of the software development process employed by the free software community in "The cathedral and the bazaar" (Raymond, 2001). He argued that the process is effective at producing superior software and considers numerous reasons that make it

effective (which we explore further in Section 10.3). It was at this time that many suggested that most free software developers were either unaware of or motivated by something other than the free software ethos promoted by Stallman. (See Bonaccorsi and Rossi, 2004 and Hertel et al., 2003 for subsequent verification of this observation.) In that same year, Perens published the Debian Social Contract to articulate the developers' commitment to open source software and its users (Perens, 2002). One thing that distinguishes the Debian Social Contract from the GPL is that the needs of the users trump the priority of software freedom as defined by the GPL. Item 4 states, "We will not object to non-free works that are intended to be used on Debian systems." It is clear that Raymond and Perens sought to shape free software into an acceptable choice for businesses by defining open source software so that there are no restrictions on distributing it with proprietary software.

The business case that Raymond and Perens first made was to Netscape, attempting to convince them to make the source code for Netscape Navigator available to the free software community and remove restrictive proprietary licensing terms. In the process it became clear that the business issue was not so much making the source code available to others, but losing control over derivative works. The GPL's copyleft prevented the business from ever "closing" the source code. Raymond, the pragmatist, was motivated by purely practical terms (the widespread distribution of source code is an effective software development technique). Because software freedom was not of particular interest to many free software developers and losing control over derivative works was a risk that business was not willing to take, the requirement to spread software freedom (the notion of copyleft) was weakened and the notion of "Open Source Software" was developed. The Debian Social Contract, which contained the Debian Free Software Guidelines, became the basis of the Open Source Definition. The OSI now publishes licenses that meet the Open Source Definition and declares software distributed under any of these licenses as "OSI Certified."

On philosophical grounds, Stallman is a most ardent critic of Open Source Software. He has two main objections. The first has to do with the weakening of the notion of Free Software. While making the source code available with the executable version will allow a user to achieve most of the four software freedoms, there are ways to license software and the source code that will allow certain users to keep their modifications private (in the sense of source code) while releasing only the executable version. Stallman sees the ability to do this as a violation of the tenets of free software. Putting the needs of any particular user/developer ahead of the concept of software freedom is unacceptable. He is also unabashed in his objection to the use of the term "open source." He argues that obscuring "free software" behind the "open source software" moniker hides the ideals that free software promotes (Stallman, 1998). When people use open source software for pragmatic reasons, there is no reason to believe that they truly understand the ethical importance of free software. Stallman believes that people who use free software and understand the social implications attached to its use and development are much more likely to include the social implications in their deliberations surrounding a switch to proprietary software. As evidence, he recounts a number of incidences where executives in the open source industry publicly indicated a lack of appreciation of the ideal of free software. He attributes this unawareness to the use

of the term "open source" rather than the use of the term "free." We do note, however, that Stallman does not object to most of the practices of the Open Source Software community. The fact that the source code on an open source project is available to all is a necessary, but not sufficient, part of software freedom.

Chopra and Dexter also offer analysis of the distinction between FS and OSS (Chopra and Dexter, 2005). They start by noting that software now plays an essential role in the social and political lives of many people and ask the question of whether (open source) software developers are morally obligated to apply copyleft to their work. After taking "as a bedrock principle that freedom is a moral good" and "[t]hat the only justifiable violation of this freedom is the restraint of a person whose actions interfere with the liberty of another," they conclude that FS is the morally superior choice to OSS. Their argument centers around four points. First, they note that the restrictions of copyleft only affect the act of distribution. That is, most freedoms are not affected by copyleft. Next they observe that copyleft does not restrict the ability of the licensee to earn a living, because someone can still be hired to make modifications to copyleft code. Although it is certainly the case that software authors can still make a living developing software, it is not the case that copyleft does not impinge on the methods that they can use to do so. The requirements of copyleft demand that source code be made available, all but ensuring that the author cannot make a living off the distribution of copylefted code. Chopra and Dexter go on to argue that there is no coercion in copyleft.[6] They state that all choices by the original developer and subsequent modifiers of the source code are made with full knowledge of the terms of the license, and, thus, all involved support the notion of free software and perpetuate that notion. Numerous studies of free software developers seem to indicate the contrary (Bonaccorsi and Rossi, 2004; Hertel et al., 2003). Most developers participate for reasons other than promoting free software. It is not unreasonable to conclude that some of them grudgingly contribute to free software. A contributor may be in a position of not wanting to give up distribution rights (as is required by copyleft), yet wanting to make the source code available. Such a contributor must choose one or the other, but not both. Finally, Chopra and Dexter state that OSS developers take the position that developing software is "just engineering" and "free software is not a social or moral imperative." They seem to discount the fact that there may be times when free software may not be worth anything to society. This may be best demonstrated by the fact that FS was largely unknown until the start of the OSS movement. Without the OSS movement, free software might not have moved into the mainstream and business would not have considered it as a viable alternative to proprietary software. By introducing the notion of OSS, the FS community is now in a position to have its ideals considered by a wider audience.

10.2.2 Critiques of Free and Open Source Software

One of the sharpest ethical attacks on free software came from Bertrand Meyer. In the essay "The ethics of free software," Meyer lumps both free software and open source

[6]We include Watson's critique of this point in the next section.

software into the same category (Meyer, 2001). Unfortunately, his analysis begins with some assumptions that are inconsistent with those of Stallman and Raymond. In particular, he assumes that software is the legitimate property of someone and that "free software" is defined in terms of being no-cost, unrestricted in its use, and freely available in terms of the source code. These assumptions make his analysis more indirect. Nonetheless, he has a number of points that require consideration. The first is his assumption that software is the legitimate property of someone. Stallman rejects this notion by arguing that the analogy between real property rights and intellectual creations is weak: "Our ideas and intuitions about property for material objects are about whether it is right to *take an object away* from someone else. They don't directly apply to *making a copy* of something" (Stallman, 1992). Yet Meyer believes we must consider the software developers who have "contribute[d] their time, energy and creativity to free software" (Meyer, 2001). Stallman does not object to the remuneration of the developers. His objection is the restrictions placed on software users because they do not have access to the source code. Meyer sees giving source code away as "an immediate business killer" (Meyer, 2001).

Meyer's second ethical critique stems from the fact that much free software is a "copycat" of some proprietary piece of software. He points out that making software that mimics proprietary software is not unethical, but failing to acknowledge the original proprietary piece of software is an ethical lapse on the part of the developer. Meyer states that "much of the hard work and creativity goes into specifying a system" and that the implementation is really not a place that brilliance is demonstrated. Because, by necessity, the interface is publicly available, it serves as a basis for competitors, both proprietary and free, to begin their work. Although such a lack of attribution may be an ethical lapse, it is not one that speaks to the ethics of free software, but to the ethics of the developer of a particular software package.

Brett Watson offers a critique of both free and open source software that takes an interesting perspective. In *Philosophies of Free Software and Intellectual Property,* he claims that when "one takes the stance that copyright is evil," leveraging copyright to promote the ethical notion of freedom is in itself unethical (Watson, 1999). In particular, Watson objects to the entire notion of copyleft. It becomes a burden, impinging on the freedom of the developer. There should be no requirement of quid pro quo; the fact that a developer is in a position to take advantage of the software written by others does not mean the same developer must return his contributions to the software development community. Watson calls copyleft a coercive system and on those grounds objects to the notion of copyleft. In some sense, he tries to take the argument to a different level. If one is truly concerned about freedom (rather than just software freedom), then one must not try to control the behavior of others. "Advocates of a non-coercive system may themselves dislike being coerced, and by application of 'the golden rule' hence refrain from coercing others" (Watson, 1999).

Watson also considers the use of copyright to promote a noble cause, in this case software freedom that is embodied by the GPL. He argues that there is no reason to limit the promotion to this one noble cause and that we might expect to see other licenses that promote additional noble causes. In fact, he suggests that the only logical conclusions are licenses that include either none or all of an author's noble causes.

Watson does acknowledge the fact that for the GPL to be effective, authors must choose to adopt it. The GPL is designed in such a way that it contains the least restrictive set of clauses needed to promote software freedom. Any more than that and it becomes controversial (as we shall see in the discussion surrounding version 3 of the GPL), and members of the free software community will not adopt it for their software.

Perhaps Watson's most insightful critique of free software is that in some sense copyleft is not really about freedom, it is more about making sure that someone does not earn money off someone else's hard work—even though that person had voluntarily and knowingly given the work away to others. (In some sense copyleft is like preventing the purchaser of a Habitat for Humanity home from selling it at a profit and keeping the money.) Watson points out that if someone takes a piece of software, modifies it, and then tries to sell it without source code, the original piece of software is still available for all to look at and use. The freedom of the original piece of software is unaltered in this process. Stallman would counter that the software modifier in this scenario has done society an injustice by not making the source code for the modified software available. He does not have any qualms with putting legal barriers in place to prevent this sort of antisocial behavior. Watson, on the contrary, sees copyleft as impinging on the autonomy of the software developer, thus reducing freedom. He suggests a scenario in which a developer has no particular attachment to the software and what becomes of it. "[I]t may be flattering to such an author that someone else wishes to create a derived work, regardless of whether that derived work will be free or proprietary" (Watson, 1999).

Watson thinks the world would be a better place if there were no copyright restrictions whatsoever. Without copyright, there is a chance that all software will truly be free—no restrictions at all. However, Watson is a realist in recognizing that this ideal will never be achieved and acknowledging that copyleft is a pragmatic way to maximize most freedoms.

10.2.3 The Controversy Regarding GPL Version 3

In 2006, the Free Software Foundation released drafts of version 3 of the GPL for commentary by the worldwide free software community. In response to technological and legal developments that have occurred since the adoption of GPLv2, GPLv3 articulates the notion of software freedom in a much more nuanced way. One of the more controversial aspects has been the language that deals with the issue of Digital Restrictions Management. The preamble of the first draft of GPLv3 clearly stated that "Digital Restrictions Management is fundamentally incompatible with the GPL" and that it "ensures that the software it covers will neither be subject to, nor subject other works to, digital restrictions from which escape is forbidden" (GPLv3,1st draft, 2006).

It is worth noting that most of the proprietary software industry uses the acronym DRM to refer to "Digital Rights Management." Digital rights management refers to software that copyright holders use to manage creative content and to control the copying of electronic versions of that material. The Free Software Foundation clearly takes a different interpretation of the matter of whose rights are being interfered with by such software. This really comes as no surprise because Stallman has been

concerned about this issue for some time. In the essay entitled *Linux, GNU, and Freedom*, he noted some troubling developments with respect to software freedom within the Linux kernel (Stallman, 2002). He stated that the Linux kernel distributions at that time contained nonfree software that probably made it illegal for them to be distributed. This nonfree software came in two forms as part of device driver software. There were special numbers that needed to be placed in the device registers by the driver and a binary form of a substantial piece of software. To deal with this threat of DRM, GPLv3[7] has a section entitled "No denying users' rights through technical measures" with the following terms:

> No covered work constitutes part of an effective technological "protection" measure under section 1201 of Title 17 of the United States Code.[8] When you convey a covered work, you waive any legal power to forbid circumvention of technical measures that include use of the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing the legal rights of third parties against the work's users.

There is another section that deals with the DRM threat by addressing the potential for DRM present in hardware rather than software. The term deals with the requirements for the distribution of source code when object code (the executable version) is the primary distribution mechanism.

> The Corresponding Source conveyed in accord with this section must be in a format that is publicly documented, with an implementation available to the public in source code form, and must require no special password or key for unpacking, reading or copying.

This part of GPLv3 would apply to software that a device (say, a cell phone) manufacturer puts on the device. It would allow the user of the apparatus (or anyone, for that matter) to obtain the source code even though it was not included in the package with the device.

While the language is still fluid as we write this, GPLv3 contains clear language regarding software users' freedoms and suggests pushing those freedoms to other realms. Many within the OSS community are ambivalent about the promotion of software freedom as expressed in GPLv2. However, many OSS developers are concerned about using a software license to push freedom in other realms. Linus Torvalds, the original developer of Linux and holder of the copyright to much of the core of Linux, is quite straightforward about his objection: "The Linux kernel is under the GPL version 2. Not anything else. . . . And quite frankly, I don't see that changing" (Torvalds, 2006a). He is particularly concerned about the first provision noted above. In reference to it, he says, "I believe that a software license should cover the software it licenses, not how it is used or abused—even if you happen to disagree with certain types of abuse" (Torvalds, 2006b). In that same post he goes on to suggest that there are

[7]All quotes from GPLv3 are taken from the second draft, the most current draft at the time of this writing.

[8]This is a reference to the Digital Millennium Copyright Act.

some uses of DRM that might be for the greater good. Curiously, Torvalds has caught Stallman in a contradiction—GPLv3 restricts the use of FS in DRM, yet Software Freedom 0 demands the freedom to run a program for *any* purpose.

The FSF is very clear about the importance of the anti-DRM clauses in GPLv3. Eben Moglen, director of the FSF, in a speech to FSF members said that the anti-DRM clauses in GPLv3 are there because the FSF is "primarily fighting to protect our way of making software" (Moglen, 2006). Because DRM technologies are often used to control access to content other than software (e.g., movies and music), there are those that view the anti-DRM clauses as an attack on the content developer's rights to control the distribution of that content. Essentially, the content developer's right to control his/her creations is "collateral damage" that the FSF is willing to accept in its promotion of software freedom. This view is consistent with Stallman's focus on the rights of all users. In the same speech Moglen says, "What we are playing for is the same thing as always: rights of users." He expresses grave concern about giving up control of many aspects of our personal lives to DRM technology as more and more of the functions within our homes are based on computing technology.

In addition to the controversy surrounding the anti-DRM terms in GPLv3, GPLv3 has raised a practical issue with ethical consequences. In addition to offering GPLv2 for others to use, the FSF offers boilerplate language to use when software is licensed under GPLv2. This language allows a redistributor of GPL'd software to license it with either version 2 or any later version of the GPL. A number of years ago (there is some controversy about exactly when and for which files), Torvalds and other major kernel developers began removing the "or later" clause from the software they wrote. Torvalds has said, "Conversion isn't going to happen" in reference to a possible conversion from GPLv2 to GPLv3 (Torvalds, 2006a). This is a clear demonstration that Torvalds' conviction to software freedom is more pragmatic than that of Stallman's, yet his strong belief in GPLv2 suggests that it is not as pragmatic as Raymond's and Perens'.

10.3 WHY OSS FLOURISHES

The social contract articulated in the Open Source Software Definition is fairly clear about what OSS offers to others. But what do OSS developers expect in return? What motivates developers to contribute to an open source project? Is it altruism, that is, 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).

The analysis of motivations of OSS developers can best be traced through the writings of Eric S. Raymond and Bruce Perens, cofounders of the OSI initiative. The OSI initiative was developed in 1998 and attained general public notice through the publication of Eric Raymond's "The cathedral and the bazaar" (1998, revised 2001), *Homesteading the Noosphere* (2000), and *The Magic Cauldron* (1999, revised 2002).

The impetus for Raymond's initial publication was the emergence of Linux, Linus Torvalds' bazaar-like operating system project. In "The cathedral and the bazaar," Raymond, skeptical that Torvalds' method would work on other large open-source projects, applied it to a project of his own. The results are detailed in the article. The next two articles are further attempts to identify the motivations of OSS developers and the economics associated with the OSS community. Perens in *The Emerging Economic Paradigm of Open Source* (revised 2006) updates the economic analysis started by Raymond in *The Magic Cauldron.* Yochai Benklar in "Coase's Penguin, or, Linux and the Nature of the Firm" is interested in the more general question of "large-scale collaborations in the digital information market that sustain themselves without reliance on traditional managerial hierarchies or markets" (Benklar, 2002).

All of these articles attempt to explain the customs and taboos of the OSS community as well as the sustainability of open source in an exchange market. Although there may be allusions to issues of moral obligation, that is, whether software developers have an obligation to make their source code available, in the explanation of customs and taboos, Raymond declares that he is presenting an economic-utility argument rather than a moral analysis. While Raymond and Perens take the approach of examining software production within the OSS community, Benklar observes the phenomena as a nonpractitioner and extends his observations to other domains.

10.3.1 The Motivations of OSS Developers

In "The cathedral and the bazaar" (1998), Raymond details how the success of GNU/Linux had changed his perception of what the open source community could accomplish.

> The fact that this bazaar style seemed to work, and work well, came as a distinct shock. As I learned my way around, I worked hard not just at individual projects, but also at trying to understand why the Linux world not only didn't fly apart in confusion but seemed to go from strength to strength at a speed barely imaginable to cathedral-builders.

Previously, Raymond envisaged large operating systems as being developed only in the cathedral style of traditional software development. Open source projects, to him, were small and fast, built on rapid prototypes. He decided to test the Linux method of development for himself by developing a POP client for e-mail. He was searching for the motivations that drew hackers into large projects as well as the sustainability of the projects.

Using Torvalds' model of "release early and often," Raymond discovered that there were several features that drew hackers to his project and why this seemingly chaotic model worked. The essay is organized around the lessons learned from the Torvalds model: Primarily, programmers join a community because there is a program that they need for their own personal use and they are willing to put it out to the OSS community at large. "When you start community building, what you need to be able to present is a plausible promise. Your program doesn't have to work particularly well. It can be crude, buggy, incomplete, and poorly documented. What it must not fail to do is

convince potential co-developers that it can be evolved into something really neat in the foreseeable future" (Raymond, 2001).

Raymond observes that programmers who participate in this bazaar style of development know the value of reusable code; can start over and throw away the first solution; can keep an open mind and find interesting projects that they want to code; treat users as codevelopers and listen to them; keep the beta test base large so that problems will find a solution; use smart data structures; and can ask a different question, or try a different approach when a wall is hit (Raymond, 2001). These lessons, verified in his own project, confirm Torvalds' methodology. The initial publication of this essay in 1997 drew criticisms that Raymond answered in the 2001 revision. Most traditionalists objected to the dynamic change of project groups in the bazaar style of development. They equated it to a lack of sustainability in the project. Raymond answered these objections by citing the development of Emacs, a GNU editing tool that sustained a unified architectural vision over 15 years (Raymond, 2001).

Ultimately, Raymond concludes that "perhaps in the end the open-source culture will triumph not because cooperation is morally right or software 'hoarding' is morally wrong (assuming you believe the latter, which neither Linus nor I do), but simply because the commercial world cannot win an evolutionary arms race with open-source communities that can put orders of magnitude more skilled time into a problem" (Raymond, 2001). Raymond emphasizes that one of the strengths of the OSS community is that programmers select projects based on interest and skills. He refines this argument in *Homesteading the Noosphere,* where he contrasts the OSS community, a gift culture that is marked by what you give away in terms of time, energy and creativity, with that of an exchange culture that is built on control of the scarcity of materials. In OSS there are always resources of machines and people, and those who try to participate in this culture understand that they are obligated to share their source code. He points out that the culture of OSS only "accepts the most talented 5% or so of the programming population" (Raymond, 2000).

In this article, Raymond explains that although members of the OSS community believe that open-source software is a good and worthy thing, the reasons for this belief vary. He asserts that there are various subcultures within the OSS community: those representing zealotry (OSS as an end in and of itself); those representing hostility to any and all commercial software companies, and any cross product of these two categories (Raymond, 2000). He cites Stallman as an example of a member of the hacker culture who is both "very zealous and very anticommercial" (Raymond, 2000). And, by extension, the FSF supports many of his beliefs. He contrasts the FSF with the pragmatists whose attitudes are only mildly anticommercial and who, in the early 1980s and 1990s, were represented by the Berkeley Unix group. The real shift in power within the hacker culture occurred with the advent of Linux in the early 1990s and the release of the Netscape Source in 1998. When the corporate world took an interest in OSS, the pragmatists became the majority of the hacker culture. By the mid-1990s, this manifested itself in programmers who identified more with Torvalds than with Stallman, and who were less zealous and hostile. The OSS community became more polycentric, developing their own non-GPL licensing schemes.

The OSS development model seems to verify that it is economical and productive to recruit volunteers from the Internet. In general, OSS developers fall into one of two broad categories: hobbyists who enjoy writing software and those who work for a corporation or agency that requires its developers to make contributions to either free or open source software projects. Regardless, there are norms dealing with taboos and ownership customs that act as moral guidelines within the OSS community. The taboos are against forking projects (breaking off and working on another version of the project), distributing changes without the approval of the project owners, and removing someone's name from the credits of a project without prior permission. Owners of OSS projects are likened to the homesteaders of the wild frontier. Homesteaders are those who assume ownership of a project by cultivating the idea and interesting the hacker community, by taking it over from another "owner" who passes it to them, or by picking up a project with no clear chain of ownership and making it their own (Raymond, 2000). In the last case, custom demands that you actively look for the owner and announce that you intend to take over the project. Raymond observes that hackers have been following these norms for years, and that they have evolved. Even in the OSS community there has been movement to "encourage more public accountability, more public notice, and more care about preserving the credits and change histories of projects in ways that (among other things) establish the legitimacy of the present owners" (Raymond, 2000).

In his Lockean analogy, Raymond suggests that the expected return from the programmer's labor comes in the form of reputation among others within the community not only as an excellent programmer, but also as a keeper of the customs associated with homesteading. The recognition of reputation can come only from those already recognized within the culture, and criticism is always directed at the project and not at the person. "The reputation incentives continue to operate whether or not a craftsman is aware of them; thus, ultimately, whether or not a hacker understands his own behavior as part of the reputation game, his behavior will be shaped by that game" (Raymond, 2000). While one could argue that reputation might translate into an economic benefit in a traditional market, what you give away leads to social status within the OSS community. "In the hacker community, one's work is one's statement. There's a very strict meritocracy (the best craftsmanship wins) and there's a strong ethos that quality should (indeed *must*) be left to speak for itself. The best brag is code that 'just works,' and that any competent programmer can see is good stuff" (Raymond, 2001). Therefore, Raymond points out that "the reputation game may provide a social context within which the joy of hacking can in fact become the individual's primary motive" (Raymond, 2000).

The *noosphere* in the essay title refers to "the territory of ideas, the space of all possible thoughts" (Raymond, 2001). Raymond is clear to point out that the noosphere is not cyberspace, where all virtual locations are "owned" by whoever owns the machines or the media. He comments that there is anger against companies, such as Microsoft, because these commercial companies restrict their source code to only their programmers, thereby reducing the noosphere available for development by and for everyone (Raymond, 2001). Within the OSS community itself, there are also certain elements that are not forthcoming with their code, and, in fact, at times will mislead

other OSS programmers. These are crackers who do not seem to respect the values and customs of the legitimate OSS community, nor feel morally obligated to participate in the gift culture that OSS embraces. Their philosophy is to hoard rather than share, and they clearly do not embrace the trust that is necessary in a peer review process. For Raymond, sharing good craftsmanship that helps people rewards the developer with personal satisfaction and extends the noosphere.

10.3.1.1 Autonomy One perceived attraction for OSS developers is the autonomy of the programmer. Although developers who embrace OSS do gain a measure of autonomy not available to those working on commercial software, the claim for *complete* autonomy does not appear to be valid. For the most part, OSS developers work as volunteers from the perspective of the project, and can join or quit an effort strictly on their own initiative. These volunteers are not coerced into participation and contribute willingly. 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 when no single developer is in charge of an OSS project. An OSS developer cannot be sure that his/her contribution will be accepted into the continuously evolving canonical version. A contribution may be embraced or rejected, and if accepted may later be changed or replaced. 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 those 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, 2001).

The developers of an open source project must take special care to avoid the symptoms of groupthink. A newcomer to open source development brings very little in terms of reputation 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 will also face 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 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 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 works to suppress this type of entrepreneurial endeavor.

10.3.2 Economic Foundations for OSS

In *The Magic Cauldron,* Raymond explores the economic foundations of OSS. He continues the discussion touched upon in *Homesteading the Noosphere* of how OSS, largely a gift culture, can economically sustain itself in an exchange economy and presents his analysis from within this context. The motivation for this essay came from the realization that most OSS developers are now working in a mixed economic context. Raymond distinguishes between the "use value" of a program, which is its economic value, and the "sale value," which is its value as a final good. He dispels the myths about the "factory model" of software, in which software is analogous to a typical manufactured good, because most software is not written for sale but rather in-house for specific environments. He states, "First, code written for sale is only the tip of the programming iceberg. In the premicrocomputer era it used to be a commonplace that 90% of all the code in the world was written in-house at banks and insurance companies. This is probably no longer the case—other industries are much more software-intensive now, and the finance industry's share of the total must have accordingly dropped—but we'll see shortly that there is empirical evidence that approximately 95% of code is still written in-house" (Raymond, 2002). Raymond examines the contradiction that software is really a service industry that is masquerading as a manufacturing industry. Consequently, he maintains that consumers lose because price structures replicate a manufacturing scenario even though they do not reflect actual development costs. In addition, vendors do not feel obligated to offer support, as their profit does not come from help center service. Open source offers an economic challenge to this model. "The effect of making software 'free,' it seems, is to force us into that service-fee-dominated world—and to expose what a relatively weak prop the sale value of the secret bits in closed-source software was all along" (Raymond, 2002). If one conceives of OSS as a service model, then consumers would benefit. Raymond cites the example of ERP (Enterprise Resource Planning) systems that base their price structure on service contracts and subscriptions and companies such as Bann and Peoplesoft, that make money from consulting fees (Raymond, 2002).

In seeking to create an economic model for OSS, Raymond tackles the notion of the commons, which at first glance might seem to apply to a cooperative community such as OSS. He rejects the model of the commons, calling OSS an inverse commons where software increases in value as users add their own features, "The grass grows taller when it's grazed upon" (Raymond, 2002). Maintenance costs and risks are distributed among the coders in the project group.

According to Raymond, sale value is the only thing threatened by a move from closed to open source. He cites two models in which developer salaries are funded out of use value: the Apache case of cost sharing and the Cisco case of risk spreading. He demonstrates that by encouraging a group of OSS programmers to work cooperatively

to build a better model in a shorter time than one could build it on one's own, companies get more economic value, and mitigate the risk of losing employees who developed the program and might change jobs. With OSS, the company now has a sustainable pool of developers. Because most of this software has no sale value, but rather supports the infrastructure of the company, OSS developers are getting paid to support use value of software. This model has emerged with the Linux for-profit companies such as Red Hat, SuSE, and Caldera (Raymond, 2002). Raymond stresses that a large payoff from open source peer review is high reliability and quality (see Section 10.3.3). Another salient point that Raymond makes is that because the shelf life of hardware is finite, support stops. For those users who continue to use the hardware, having access to the source code makes their lives a lot easier. In a sense, he says, you are "future-proofing" by using OSS.

Bruce Perens takes up the discussion of the economics of OSS in his article *The Emerging Economic Paradigm of Open Source*. Perens examines three paradigms of economic development: the retail paradigm, in which the developer hopes to recover costs from the sale of the finished product; the in-house or contract paradigm, in which programmers are paid for creating custom software; and the open source paradigm. As an advocate for the third paradigm, Perens points out the weaknesses in the other two. For retail, he states that because of its low efficiency (funding software development via retail software purchases is lower than 5%) this paradigm can only be economically feasible if products are developed for a mass market. Whereas the second paradigm is more efficient (50–80%) as it directs "most of each dollar spent toward software development," this software only has a success rate of 50% because the software often fails to meet the customer's goals (Perens, 2006).

These weaknesses disappear with the Open Source paradigm, as contributors are developing a useful product that companies or individuals need. Like Raymond, he emphasizes that the remuneration to the open source software developer may not be as direct as that of the commercial developer; yet, because more than 70% of software is developed as service for customers, there is still a monetary return for OSS programmers (Perens, 2006). Perens distinguishes between technology that makes a company product more desirable to its customers (differentiating) and technology that supports the infrastructure and is general enough so that competitors can know about it (nondifferentiating). Most software is nondifferentiating. Thus, for companies that need software to support their infrastructure, and when that software does not differentiate the business, the OSS community offers reliability, peer review, and sustainability. If a company is not large enough or does not have enough experience to develop software competitively, then open source is a smart alternative. Open source programmers are finding that, more than a hobby, they can get paid by companies such as Red Hat, O'Reilly, and VA Linux Systems to work full time on open source projects. An additional bonus is the absence of advertising costs in OSS software. "The major expense is the time-cost of employee participation" in mature OSS projects (Perens, 2006). In addition, new and creative additions to the software are constantly being developed. Perens describes various ways of using open source within a company: GNU/Linux distribution companies, companies that develop a single open source program as their main product such as MySQL, hardware vendors such as IBM and HP,

end-user businesses such as e-Bay, the government, and academics (see Perens). Perens asserts that OSS is self-sustaining because "[i]t is funded directly or indirectly as a cost-center item by the companies that need it" (Perens, 2006). Both Raymond and Perens conclude that companies will be willing to pay for the creation of open source for nondifferentiating software. If a company wishes to produce a retail piece of software, then the OSS paradigm will not work. Because the majority of software is nondifferentiating, there will be opportunity for open source collaboration.

10.3.3 The Quality of OSS

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 they are developing software for their own use; their reputations among their peers are at stake. Critics of OSS claim that volunteers will not do professional-quality work if there is no monetary compensation. This has become a rather outdated argument. As we have seen above, there are many who are employed by companies to write open source code and others who are paid to customize it. Critics also claim that documentation and maintenance are nonexistent. Although it is true that documentation and maintenance are concerns, OSS advocates maintain 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 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 in that market with high-quality software. Yet, despite commercial alternatives, the OSS Apache server is by far the most used web server. Since August 2002, regular surveys have demonstrated that over 60% of web servers on the Internet are Apache (Netcraft, 2006). At least in this market segment, it appears that OSS is sufficiently and consistently high quality for many users. Of course, Apache is free and other servers are not; 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? 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.

Another piece of evidence is a study by Coverity, a company whose software is used to detect numerous types of known software defects (Chelf, 2006). In March 2006 the

company released a report describing its results of analyzing 32 OSS packages. The defect rate ranged from 0.051 to 1.237 (defects per 1000 lines of code) with an average of 0.434. The defect rate in the better-known packages (Linux, Apache, MySQL, Perl, Python, and PHP) was an even lower at 0.290. Coverity also published the defects that it found, and in a month over 1000 of the initial 7500 defects had been fixed. Since that time, 17 more OSS packages have been added to the analysis, and of the 49 total packages, 11 have none of the defects for which Coverity searches and the remaining 38 packages have an average defect rate of 0.232 (Accelerating, 2006). This is evidence that some OSS developers take code quality seriously and strive to improve it.

The nature of proprietary software makes a fair comparison difficult. Published defect rates for commercial software vary widely, anywhere from 1 to 30, but it seems safe to say that the evidence suggests that at least these popular OSS packages have defect rates that are on par with their commercial counterparts. An earlier study that compared three unnamed proprietary software packages to Linux, Apache, and gcc (the GNU Complier Collection) concluded that the open source projects "generally have fewer defects than closed source projects, as defects are found and fixed more rapidly" (Paulson et al., 2004). A final piece of corroborating evidence is that even though Coverity offers a free analysis for proprietary code that competes with any of the OSS projects, no similar results are available.

10.3.4 The Ethical Responsibilities of Software Developers

Both open source and proprietary developers share the professional ethical responsibility to develop solid, well-tested code. However, the influences on open source software developers to maintain this ethic differ substantially. Proprietary software establishes a strong distinction between developers and consumers. An interesting aspect of OSS is that this distinction can be less pronounced, suggesting that ethical models for analyzing that relationship need to be different.

Most obviously, when developers and users of OSS neither get nor give payment, financial self-interest is no longer a major concern. Developers are not "using" consumers to get their money. Users are not trying to negotiate an unfair deal for software. Instead, both developers and consumers in OSS are cooperating freely in the OSS project.

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 contrary, 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. Although there is a reputation risk in doing so, there is the opportunity to publicly demonstrate that the forked product is superior.

Because a developer (or group of developers) typically runs an OSS project and is responsible for making decisions about the design of the software and the quality of the code, he/she is ultimately responsible for the "penumbra"[9] (all people who are under the

[9]In the case of proprietary software, software developers and others in the corporate structure share the burden of care for users and the penumbra. In the case of OSS, that responsibility falls entirely upon the software developers.

influence of a piece of software whether they realize it or not (Collins et al., 1994)). Curiously, the interests of the penumbra are closely tied to the life of an OSS project. When a project is in its early stages, the initial users are often the developers and they may be more tolerant of glitches and defects than is acceptable for the penumbra. However, as the project matures, its longevity becomes closely tied to quality. If the quality is not high enough, the project will likely terminate quickly because without any marketing money behind the project, it will not develop the strong user support it takes to make an OSS project successful. As an OSS project increases its market share, those OSS developers are increasingly obligated to consider their responsibilities to the people who use and are affected by the software. A critique of OSS is that sometimes OSS developers have pointed to the low price and claimed, "you get what you pay for" when the software is unreliable. Obviously, the ethical principle of consideration of the public good is clear: OSS developers have professional responsibilities, even though they are different from traditional professionals in how their work is rewarded.

OSS developers have a built-in "informed consent" advantage: by definition, OSS gives users the freedom to examine the source code of the application. Although the source code may only be understandable to some OSS users, this transparency of code (rare in commercial projects) is a fundamentally open stance that encourages a trust relationship between developers and users. OSS literature advocates a level of cooperation and "community" for OSS participants that is not encouraged or observed in, for example, users of shrink-wrapped commercial applications. Thus, the relationship between the developers and the users in OSS is best modeled as a trust relationship between two overlapping groups: the OSS developers on the one hand and the OSS users on the other. Trust is built in two ways. As the user base grows, nonusers can trust that the large group of users who find the software to be of sufficient quality suggests that the software is worthy of consideration. Trust is also built from the lack of financial coercion on either side. Users can explore the software without paying for it; if the users don't find the software reliable, they can choose not to use it without any financial loss. It is in the best interest of the developer to create reliable code in part to sustain his/her reputation within the open source community (see Grodzinsky et al., 2003) and in the user community. Thus, both groups gain when reliability increases and when the groups cooperate in improving the software.

10.3.5 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) defects, (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). Both (3) and (4) are general issues. Number (4) is interesting because almost all software disclaims any warranties. However, OSS does have the advantage of informed consent mentioned above.

"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; that is, cause and intent do not converge" (Johnson and Nissenbaum, 1995). When a developer contributes irresponsible code to an open source project, however, it is unlikely to be accepted. In addition, current best open source development practices attribute code to specific authors. So, there is built-in individual accountability for each code segment and the overall software package. Therefore, the many hands problem 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 defects as a software fact of life raises accountability issues. "If bugs are inevitable, then how can we hold programmers accountable for them?" she asks. The open source approach to software development treats the defect problem with a group effort to detect and fix problems. The person who finds a defect in OSS may not be the person to fix it. Because many adept developers examine OSS code, defects are found and corrected more quickly than in a development effort in which only a few developers see the code (Paulson et al., 2004). In this group effort, accountability is not lost in the group, but is instead taken up by the entire group. The question of whether or not this group accountability is as effective as individual responsibility is, again, empirical. The Coverity study offers strong anecdotal evidence that some OSS developers take defects seriously and work diligently to remove them (Chelf, 2006).

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.[10]

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, vendors slip quietly away.

[10]Curiously, GPLv3 deals with this issue head-on for a single objectionable use: DRM.

10.4 IS OSS A PUBLIC GOOD?

Stallman (2001) notes that the notion of copyright was developed in response to the (corporate) ability to mass-produce creative works, and societies establish copyright laws to promote the production of creative works. He argues that the notion of free software, and by extension open source software, is a return to the pre-printing press days when anyone (with time) could make a copy of a book. A similar notion of freely sharing ideas has also persisted in academia as well. 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 Kollock, a sociologist at the University of California at Los Angeles, examines the idea of online public goods in his paper entitled "The economies of online cooperation: gifts and public goods in cyberspace" (Kollock, 1999). He defines public goods as those things that are nonexcludable and indivisible. Because the Open Source Definition prohibits discrimination against persons or groups or against fields of endeavor, it supports the definition of a public good being nonexcludable. 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 GNU/Linux, for example, she does not diminish its availability for other users. So by this definition, we argue that OSS is a public good.

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 toward existing commercial software developers. Bertrand Meyer recites with dismay the many negative statements by OSS advocates about commercial software development and developers (Meyer, 2001). Some see "Microsoft bashing" as a central theme of the OSS movement. Because most Microsoft products compete directly with OSS packages, 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 computer users. Because the OSS movement is,

by design, decentralized and evolving, it seems impossible to gauge with any precision the motivations of all its members (Hertel et al., 2003). 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. GNU/Linux is appearing as the operating system of choice, often replacing Solaris. As more and more applications run on GNU/Linux, universities will have less incentive to buy from Unix platform vendors. They will buy cheaper hardware and run GNU/Linux. One caveat to this scenario is the availability of staff who can support the GNU/Linux platform and the availability of documentation for OSS.

Service learning, a concept that is becoming part of the mission of many higher education institutions, also influences the choice between open source software and proprietary software. Consider a scenario in which 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? Some ongoing support to these charities might be an opportunity for the university to openly support OSS as a public good.

10.5 CONCLUSION

The distinction between Free Software and Open Source Software has had a positive effect on the software development community and on the larger online community as well. Regardless of the motivation of individual developers, it is difficult to find fault with their willingness to give their creative contributions to the world to study and adapt as the world sees fit. Stallman's increasingly clear focus on freedom for all users of software and hardware has forced discussion on issues that many people today have not considered. Elevating discussion of the social purpose of copyright to an international level is valuable. Raymond and Perens' ability to articulate the necessary and sufficient aspects of software freedom that contribute to developing quality

software has been an important part of improving the quality of software that society uses. There is some suggestion that, regardless of whether the quality of FS and OSS is high, the mere possibility that it is higher than that of some proprietary software has prompted some proprietary software developers to adopt techniques and processes that lead to better software, again benefiting everyone. Finally, the Free Software movement can be credited with providing an impetus for establishing notions of freedom for other types of digital media, such as the Creative Commons (creativecommons.org).

ACKNOWLEDGMENTS

In composing this chapter, we drew some material from Grodzinsky, F.S., Miller, K., and Wolf, M.J. (2003), "Ethical issues in open source software," *Journal of Information, Communication and Ethics in Society*, I(4), 193–205, Troubadour Publishing, London, and our paper (2006), "Good/fast/cheap: contexts, relationships and professional responsibility during software development," *Proceedings of the Symposium of Applied Computing;* 2006 April.

REFERENCES

Accelerating open source quality. Available at http://scan.coverity.com/. Accessed 2006 July 24.

Benklar, Y. (2002). Coase's Penguin, or, Linux and the Nature of the Firm. *Yale Law Journal,* 112, 369–466.

Bonaccorsi, A. and Rossi, C. (2004). Altruistic individuals, selfish firms? The structure of motivation in Open Source Software. *First Monday,* 9(1). Available at http://firstmonday.org/issues/issue9_1/bonaccorsi/index.html. Accessed 2006 July 25.

Chelf, B. (2006). Measuring software quality: a measure of open source software. Available at http://www.coverity.com/library/index.html, registration required. Accessed 2006 July 24.

Chopra, S. and Dexter, S. (2005). A comparative ethical assessment of free software licensing schemes. *Proceedings of the Sixth International Conference of Computer Ethics: Philosophical Enquiry (CEPE2005)*, Enschede, The Netherlands, July.

Collins, W.R., Miller, K., Spielman, B. and Wherry, P. (1994). How good is good enough? An ethical analysis of software construction and use. *Communications of the ACM*, 37(1), 81–91.

Free Software Foundation (1991). GNU general public license. Available at http://www.gnu.org/licenses/gpl.txt. Accessed 2006 July 19.

GPLv3 1st discussion draft (2006). Available at http://gplv3.fsf.org/gpl-draft-2006-01-16.html. Accessed 2006 August 7.

GPLv3 2nd discussion draft (2006). Available at http://gplv3.fsf.org/gpl-draft-2006-07-27.html. Accessed 2006 August 7.

Grodzinsky, F.S., Miller K., and Wolf, M.J. (2003). Ethical issues in open source software. *Journal of Information, Communication and Ethics in Society,* I(4), 193–205.

Hertel, G., Neider, S., and Herrmann, S. (2003). Motivation of software developers in open source projects: an Internet-based survey of contributors to the Linux kernel. *Research Policy,* 32, 1159–1177.

Himma, K.E. (2006). Justifying intellectual property protection: why the interests of content-creators usually wins over everyone else's. In: Rooksby, E. and Weckert, J. (Eds.), *Information Technology and Social Justice*, Idea Group, pp. 54–64.

Himma, K.E. (2008). The justification of intellectual property rights: contemporary philosophical disputes. Perspectives on Global Information Ethics. *Journal of the American Society for Information Science and Technology,* 59(7).

Johnson, D.J. and Nissenbaum, H. (Eds.) (1995). *Computers, Ethics and Social Values*. Prentice Hall, New Jersey.

Kollock, P. (1999). The economies of online cooperation: gifts and public goods in cyberspace. In: Smith, M. and Kollock, P. (Eds.) (1999), *Communities in Cyberspace*. Routledge, London.

Meyer, B. (2001). The ethics of free software. *Dr. Dobb's Portal*. Available at http://www.ddj.com/dept/architect/184414581. Acessed 2006 July 19.

Moglen, E. (2006). The hardware wars. *The Free Software Foundation Bulletin,* 8 June.

Netcraft (2006). July 2006 Web server survey. Available at http://news.netcraft.com/archives/web_server_survey.html. Accessed 2006 July 24.

Nissenbaum, H. (1994). Computing and accountability. *Communications of the ACM,* 37(1), 72–80.

Paulson, J., Succi, G., and Eberlein, A. (2004). An empirical study of open-source and closed-source software products. *IEEE Transactions on Software Engineering,* 30(4), 246–256.

Perens, B. (2002). *Debian Social Contract*. Available at www.debian.org/social_contract.html. Accessed 2006 July 19.

Perens, B. (2006). *The Emerging Economic Paradigm of Open Source*. Available at http://perens.com/Articles/Economic.html. Accessed 2006 March 1.

Raymond, E.S. (2000). *Homesteading the Noosphere*. Available at http://www.catb.org/esr/writings/homesteading/homesteading/. Accessed 2006 July 19.

Raymond, E.S. (2001). The cathedral and the bazaar. In: Spinello and Tavani (Eds.), *Readings in Cyberethics*. Jones and Bartlett, Sudbury, MA.

Raymond, E.S. (2002). *The Magic Cauldron*, Version 2.0. Available at http://www.catb.org/~esr/writings/cathedral-bazaar/magic-caulddron/index.html. Accessed 2006 June 06.

Schach, S. (2002). *Object Oriented and Classical Software Engineering*, 5th edition. McGraw Hill, p. 137.

Stallman, R. (1985). *The GNU Manifesto*. Available at http://www.gnu.org/gnu/manifesto.html. Accessed 2006 July 19.

Stallman, R. (1992). *Why Software Should Be Free*. Available at http://www.gnu.org/philosophy/shouldbefree.html. Accessed 2006 July 19.

Stallman, R. (1994). *Why Software Should Not Have Owners*. Available at http://www.gnu.org/philosophy/why-free.html. Accessed 2006 July 19.

Stallman, R. (1998). *Why "Free Software" is Better than "Open Source"*. Available at http://www.gnu.org/philosophy/free-software-for-freedom.html. Accessed 2006 July 19.

Stallman, R. (2001). Copyright versus community in the age of computer networks. Available at http://www.gnu.org/philosophy/copyright-versus-community.html. Accessed 2006 August 8.

Stallman, R. (2002). *Linux, GNU, and Freedom*. Available at http://www.gnu.org/philosophy/linux-gnu-freedom.html. Accessed 2006 July 19.

The Open Source Definition (2006). Version 1.9. Available at http://www.opensource.org/docs/definition.php. Accessed 2006 July 20.

Torvalds, L. (2006a). Re: Linux vs. GPL v3—dead copyright holders. *Linux-kernel mail archives*, January 25, 2006. Available at http://www.ussg.iu.edu/hypermail/linux/kernel/0601.3/0559.html. Accessed 2006 July 20.

Torvalds, L. (2006b). Re: Linux vs. GPL v3—dead copyright holders. *Linux-kernel mail archives*, January 27, 2006. Available at http://www.ussg.iu.edu/hypermail/linux/kernel/0601.3/1489.html. Accessed 2006 July 20.

Watson, B. (1999). *Philosophies of Free Software and Intellectual Property*. Available at http://www.ram.org/ramblings/philosophy/fmp/free-software-philosophy.html. Accessed 2006 July 19.

Wolf, M.J., Bowyer, K., Gotterbarn, D., and Miller, K. (2002). Open source software: intellectual challenges to the status quo. Panel presentation at 2002 SIGCSE Technical Symposium, *SIGCSE Bulletin*, 34(1), 317–318. Available at www.cstc.org/data/resources/254/wholething.pdf.