Society and open source

Why open source software is better for society than proprietary closed source software

Ben Pfaff
Ken David, Ph.D., M.B.A.

1 Open source software definition

> One Saturday afternoon I bought a radio card for my computer, that is, an internal radio receiver. I selected a particular model because I knew that it was compatible with my computer's operating system, which is GNU/Linux. However, when I got it home and installed it in my computer, I found out that it was an incompatible, revised version, which meant that there was no way for me to use it without writing my own driver for it.
>
> I could have returned the card, but that's not my style. Instead, I spent the weekend reverse-engineering the card itself and its Windows 95-based driver software. On Sunday night I had a simple program that would tune arbitrary FM radio stations. By Monday morning I had written a Linux kernel driver for the card and submitted it to the Linux kernel developers. Within a week, my driver had been integrated into the development version of the Linux kernel.
>
> —an open source programmer

Proprietary closed source software is something that many of us have to deal with on a day-to-day basis. Microsoft, Netscape, Lotus, Corel, and all the rest of the big software houses are in the proprietary closed source software business.

We're all familiar with how proprietary software works: in return for a fee,[1] you receive a "license" that permits you to run the software on exactly one computer. You are forbidden from copying the software or sharing it with others. You cannot change how the software works (in order to fix bugs or add features) because the software is provided in a fixed, immutable form.

Commercial open source[2] software is a different model. As before, you pay a fee[3] and in return receive a copy of the software. However, you are not forbidden from copying the software or sharing it with others. Additionally, you are provided with "source code" to the software, which allows you to modify the software in order to fix bugs, add features, or integrate it with other software.

The following sections will show why commercial open source software is better for society than proprietary closed source software.

[1] Some proprietary closed source software is offered for no fee, i.e., Microsoft Internet Explorer. This changes none of the basic principles.

[2] The phrase "open source" is a trademark of Software in the Public Interest.

[3] Often, open source software can be obtained without paying a fee. "Open" refers to the right to change and share software, not to cost.

2 Setting and task environment

2.1 Closed source

Historically, the Internet Engineering Task Force (IETF) has crafted the protocols used for communicating on the Internet. The IETF designed the basic protocols, conventions for communication among computers, used on the Internet. These protocols work well due to the careful design and testing process employed by the IETF. Each proposal must pass through this process before being approved as a standard. The process ensures that IETF standards are clear and unambiguous.

Before approval as a standard, there must be at least two different implementations, or programs that make use of the protocol, generally developed by different organizations, that are able to work together. Although IETF standards do not require that either implementation be open source software, it would be difficult to test any other type of implementation to the IETF's satisfaction.

In recent years, many proprietary software companies have entered the Internet arena. Along the way, they've invented new protocols without passing them through the IETF standards process or any comparable standardization process. Among the results is the famous incompatibility between Microsoft's and Netscape's competing web browser programs.

This incompatibility is only one symptom of a larger problem that proprietary software poses. The computer industry deals with largely standardized products. In order to distinguish themselves, each proprietary software company adds features to its implementation of a standard. Less ambitious changes may allow compatibility to be retained, but larger changes must cause compatibility to broken. Unfortunately, larger changes are more attractive to a company because they distinguish its product more than minor changes.

2.1.1 Pressure against compatibility

Compatibility with standards is not in the best interest of a proprietary software company. Compatibility allows users to change software providers in the future, eliminating a source of revenue for the company.

Even better for a company's purposes than simply modifying an established standard is to invent a completely new "standard," without publishing information on it, in order to prevent competitors from learning how it works.[1] In this way, users are locked into a particular product from one company. When programs are also designed to work better with the same company's other products than with products from competing firms, users can be locked into using that company's entire line of programs for the foreseeable future.

A subtler variation on not publishing information on a protocol or product's workings is to publish vague, incomplete, misleading, or simply incorrect information.[2] This tac-

[1] This is commonly known in the computer industry as the "not invented here" syndrome.

[2] One other approach is to make information available only for a large fee, or only to selected vendors. This brings up the issue of open standards, an issue separate from open software but related to it.

tic allows a company to maintain the guise of using an open standards process, but still effectively prevents the development of compatible or competing products.

In addition, by being incompatible, a need for (profitable) technical support is created. See Chapter 4 [Support], page 10, for more details.

2.1.2 Divergence for commercial purposes

There are a number of projects seeking to develop clones of Microsoft's Windows operating system. This is proving difficult because so much of Windows' programming interface is undocumented. Other parts are poorly documented or incorrectly documented. For more information see Schulman [1992].

Companies seeking to defend such practices often use words like "innovation." This is clever deception, since "innovation" is a word with positive connotations meaning "new." A better word than "innovation," would be one with a negative connotation such as "imposture," since "innovation" of this sort does little other than entrap users.

In short, a proprietary software company that seeks to maximize its profits does so at the expense of the public. The additional features added by the company's product, whether by extensions to an existing standard or definition of a proprietary new "standard," would be better added by proposing extensions to an existing standard or proposing a new standard through a standards organization, such as IETF, IEEE, POSIX, W3C, or ANSI/ISO. In any case, any benefits to consumers are outweighed by their loss of freedom to change vendors and work with other vendors' programs.

In effect, a proprietary software company that adds "features" to a standard, creating a "innovated" standard, is in a monopoly position. Lack of documentation form an artificial barrier to entry, preventing other companies from producing a compatible product.

2.2 Open source

Commercial open source software, on the other hand, does not and in fact *cannot* suffer from the same problems regarding compatibility with standards. To understand why, consider the incentives faced by commercial open source developers. These developers know that they must be able to meet their development costs with just a few software sales, since users are allowed to redistribute the software they buy.

For this reason, the development of open source software is most often user-driven. A group of potential users band together to sponsor development of a new product or product feature. It is rare for an open source software company to pick a target market in hopes of attracting enough sales to pay for development.

2.2.1 No proprietary standards

Because of the limited number of potential sales, an open source company has no incentive to lock its users into proprietary "standards." Without a reason to avoid compatibility, most companies would naturally use an established, well-defined, well-thought-out standard: indeed, most important open source software adheres closely to standards.

When, for whatever reason, an existing piece of open source software does *not* fully conform to existing standards, another aspect of open source software becomes important. Since every user has access to source code, the program can be modified to make it fully compliant with the standard. The modified, compliant version is then likely to supplant the original version.

For the most part, computer standards are not legislated by any governmental or authoritative body. Instead, standards are established either through widespread use (a *de facto* standard), or by the establishment of a standard by a standards organizations such as one of those mentioned above. It should be emphasized that these organizations have little or no real power; their standards can be and sometimes are ignored.

Suppose that an open source author decides to invent a completely new non-standard protocol, imitating colleagues in the proprietary software industry. One possible result is again that the software will be changed to adhere to an existing standard. But suppose further that this new creation is truly superior to whatever standard came before it. In this case, the new protocol or format may become a new *de facto* standard. With open source software, the issue of an undocumented protocol cannot arise—the source code to a program implementing a protocol is a well-defined specification for the part of the protocol that the program implements.

So, every open source program either implements an existing standard or it unambiguously defines a new standard. Sometimes a formal standard document is then written as a form of refinement, but as often the source code itself remains the best definition. This is why open source software cannot fail to conform to a standard—it constitutes in itself the definition of a standard.

2.2.2 Pressure for innovation

Innovation in open source software can be innovation in a true sense. Instead of being intended to force users into a particular product, changes are often intended to increase compatibility with other products. "Features" that prevent interoperability can be turned off or they are never written because there is no incentive to write them.

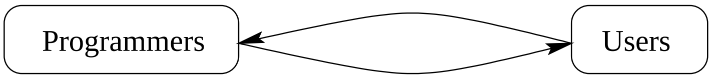
2.3 Social impact

For these reasons, open source software is targeted at the needs of its users and developers, rather than the bottom line of the company that originally writes it. Compatibility is just one way, among several, that open source software provides social benefits to a wider slice of society than does proprietary closed source software.

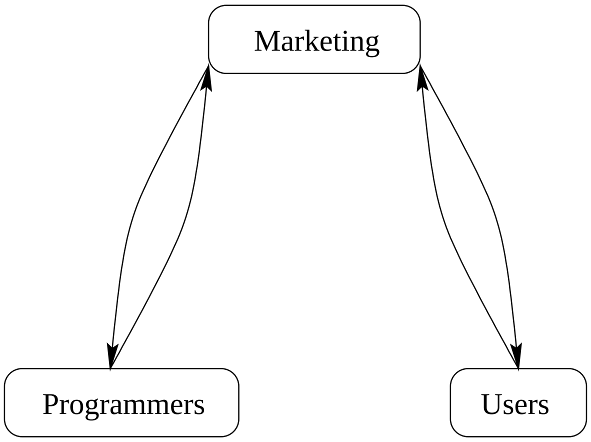
In addition to the business logic reasons outlined above, there are social reasons why open source software is more compatible than closed source software. Let's take a closer look.

For any software program, there are users and programmers. Programmers design and implement the program's internals. They may also design the program user interface and test the software before release. Users, on the other hand, use the program without necessarily understanding its internals. They may also find bugs and report them to the programmers.

The most natural relationship between users and programmers is directly reciprocal. Programmers write software for users; users report bugs and suggest features to programmers. This relationship is enhanced when programmers are also users, since then programmers better understand their users' needs. This is a common situation in open source software, illustrated below.



Proprietary closed source software development encourages partitioning of these groups. A barrier is forced between users and programmers: the closed source company's marketing and public relations department. The revised relationships are illustrated below.



The consequences of this forced barrier are socially disturbing. Now interaction between programmers and users must pass through an intermediate step. This barrier to communication causes distortion by itself. Additionally, the intermediaries have an agenda of their own, different from the needs of users and programmers, so they intentionally do not relay information back and forth in an ideal manner.

The social structure of proprietary closed source software does not encourage programmers to become users. As a rule, people are more interested in what they have a personal stake in. Closed source programmers are less likely to have a personal stake in a program; usually, they write code because they are paid to write it, not because it is software that they want to or need to use.

Inserting an intermediary between users and programmers also prevents the formation of a community consisting of both users and programmers. See Section 4.5 [Social Impact (Support)], page 12, for more details.

3 Towards effectiveness of testing

3.1 Closed source

Encryption is concealing the contents of a message or other information using a secret code. These days, algorithms (formally specified methods of performing a task) for encryption are executed by computers using complicated mathematical techniques. Encryption algorithms are designed by mathematicians.

Algorithms for encryption are rarely kept secret for a very good reason: no one person or group can be sure that any encryption algorithm is "secure," meaning that there is no easier way to crack it than to try all the possible codes. Thus, anyone who knows this will publish his or her new encryption algorithm and let the encryption community point out problems. This is a form of peer review, employed through the academic and scientific community.

Someone new to encryption will typically introduce a new software system for encryption without publishing the algorithm. Historically, these systems have proven easy to crack even without knowledge of the algorithm because their inventors overlooked some fundamental flaw in the algorithm. This could not have happened if their work had first been scrutinized by the encryption community. This sort of false sense of security obtained by witholding information is called "security through obscurity."

There is an analogous issue with proprietary closed source software. Such software is developed by a single person or company and its source code hidden from the public. As a result, only a few people ever see the program's internal workings. Fewer people examining how the program works means that fewer bugs in the program will be found and fixed. The final result is that the program may be buggy.

3.1.1 Ineffective beta testing

Many proprietary closed source programs pass through a period of beta testing:[1] before final release. However, this beta testing tends to be ineffectual at correcting remaining bugs. One reason is that the team that developed the software originally is the same one attempting to fix bugs. As experienced programmers know, the subtlest bugs cannot be easily found by a program's own author because he or she has too many built-in assumptions about the code.

Industry pundits recognize this problem of ineffective beta testing, although not its cause or its proper solution. When a major new version of a popular proprietary closed source program is released, their recommendation is to avoid upgrading until it becomes unavoidable. Their stated reasoning is that upgrading may cause unanticipated problems. In other words, "We don't know how to work around this version's bugs yet, so watch out!"

[1] *beta test*: a possibly buggy program is released to a limited group outside the company developing it, in an attempt to ferret out remaining bugs. In proprietary software beta testers must sometimes pay for this "privilege"!

Another reason beta testing tends to be ineffective is the fixed schedules imposed on the software developers. The proprietary software company probably announced that the software would be released on a particular date. This places an onus on the software developers, forcing them to release software that's possibly not ready simply because the schedule demands it.

3.1.2 Deniability

Companies do not want to publicize the fact that their software has bugs. This, in turn, tends to lead to a policy of denial, in which the company does not admit the existence of bugs in its software. Even when bugs are fixed in a later release, it is rare to see a list of exactly *which* bugs were fixed. Bug-fix releases are poorly publicized when they occur, since publicity conflicts with the denial policy, and, as a result, few users update to the new release. This may be a good thing, since these bug-fix releases necessarily receive little testing and thus often introduce a new set of bugs.

A policy of denial causes deeper problems as well. If a company would acknowledge bugs and correlate reports, then send test bug-fixes to customers with problems, or if it would put customers with similar problems in touch and allow them to work together to find the underlying cause or a way to avoid the problem, many more bugs might be acceptably fixed. But companies do not do this because of their denial policies.

3.1.3 Deflection

Finger-pointing is also a problem in the proprietary software industry. When two companies' software are installed on a user's system, and a problem occurs, Company A often blames the problem on Company B's software, whereas in turn Company B blames the problem on Company A's software. Most of the time the companies involved are more interested in fixing blame than fixing the problem, so the bug is never fixed.

3.1.4 Technical support business logic

Technical support is another reason that proprietary closed source software is excessively buggy. Many proprietary software companies make a significant amount of money from technical support. Many charge $15 or more for a single brief support call covering one incident. With this in mind, consider that a company can actually make more money selling buggy software by profiting from technical support. Even if companies do not actually sell buggy software intentionally, this does not encourage them to fix bugs promptly.

In short, it is in the best interests of proprietary closed source software companies to sell buggy software, to ignore bug reports and deny the existence of bugs, in order to make money selling technical support to the same users who paid for its buggy software to begin with. Of course, users have no other recourse since the company that writes the closed source software is the only company that can fix the bug.

Some companies also make money from selling bug fixes. For instance, the author of this article once paid over $100 for a one-user license of a particular C++ compiler, version 5.0. Within a week, a bug-fix release, version 5.02, was made available. It cost $15 plus shipping to receive the upgrade, which included no new features.

3.2 Open source

Open source software does not have the defect problems that plague proprietary software. First of all, the source code is publicly released or maybe even developed publicly. This means that potentially thousands of people can pore over the program's internals, noticing bugs and potential problems. Such behavior has actually been observed; see Raymond [1997a] for further description.

3.2.1 Effective beta testing

It is also customary for open source software to pass through a period of beta testing. Beta test releases in open source software tend to be much more effective than in proprietary software. First, many more people are looking at the source code than had been before beta test, so bugs that the original developers were unable to see are quickly revealed.

Second, traditionally, open source software is not developed on a fixed schedule. It is released if and when it is ready, which generally means that there has been an honest attempt to identify unknown bugs and that all known bugs have been fixed. In fact, many popular open source programs remain in pre-release form, with version numbers below 1.0, for years before their developers judge them ready for final release. During this beta test period, the programs are still often used by many thousands of users.

Because open source software programs are released when they are ready, not when the schedule demands it, they rarely cause problems for users who install the latest stable version when it is first released. Unlike in the proprietary software industry, many users automatically upgrade to the latest version of a program when it is released.[2] There is very little of the "fear of X.0" seen in proprietary software. Ishee [1998] has additional explanation why proprietary software has more problems in initial releases of new versions.

3.2.2 Accountability

In open source software, no one receives benefit from attempting to conceal bugs in software. Since the source code is available, there is, in fact, no way that the existence of a bug can be denied. For this reason, each program generally comes with a detailed list of known bugs and workarounds.

In addition, many projects maintain a database of bug reports, which often includes detailed information on each bug and information on what has been done so far. These databases are usually accessible to the public through a web or email-based interface. These bug databases are invaluable for letting users and developers track subtle bugs, enabling them to eventually be fixed. In addition, all but the smallest projects have email mailing lists associated with them. Users or developers who find themselves bit by bugs can send email to all the subscribed users of a program.

Finger-pointing is rare in open source software. Since the source code is available for the programs in question, there can be little doubt where the problem lies. Sometimes the

[2] In addition, the tools available make it very easy to return to an earlier version if necessary, a matter of seconds, unlike using the complicated and slow installation and uninstallation programs common in the proprietary software industry.

proper solution can be disputed, but it is normally possible to build a workaround while the correct solution is being debated.

3.2.3 Technical support business logic

There are companies and individuals who provide paid technical support for open source software. However, since anyone can modify an open source program, these companies are not in a position to prevent the correction of bugs that are found while providing technical support.

For these reasons, open source software naturally has fewer bugs than proprietary software. This higher quality is one more reason why open source software is better for society than proprietary closed-source software.

3.3 Social impact

> You use a Windows machine and the golden rule is: Save, and save often. It's scary how people have grown used to the idea that computers are unreliable when it is not the computer at all—it's the operating system that just doesn't cut it.
>
> —Linus Torvalds
> quoted by *The New York Times*

One social impact of effective software testing is that users come to expect software to be reliable. Contrariwise, when software is unreliable, users come to expect software to be unreliable.

These expectations in turn feed back to the software publisher. In the case of reliable, open source software, the programmers hear directly about problems with the software; they in turn maintain their standards.

With proprietary closed source software, marketing places a lower priority on reliability, since users don't have expectations as high as open source software users. In turn, users receive a newer version of the closed source software in question, which is of lowered reliability since marketing didn't emphasize reliability (possibly due to time pressure to ship before competitors, as discussed above). This forms a endless cycle of unreliable software and low expectations.

4 Support

When a small- to medium-size company needs to have computers installed and maintained, it contracts with a computer consulting firm for that purpose. The consultant meets with company management to determine what hardware and software best suits the company's current and future needs. Hardware and software are obtained and installed by the consultant. Then, as problems arise and needs change, the consultant maintains and updates the systems.

In a typical office network designed by a consultant, user machines run Windows 95 or Windows NT Workstation. The network servers run Windows NT Server, NetWare, or some flavor of UNIX, such as Sun Solaris. These are all proprietary closed source operating systems, designed to primarily run proprietary closed source software.

More and more consultants are moving open source software into companies and providing commercial support for their use. For the consultant, this is good news: as established earlier, open source software has fewer bugs and more compatibility than proprietary software, so setup and maintenance are easier and faster. This is good news for the client company as well, since less time for setup and maintenance means lowered costs.

4.1 Servers

Servers, computers that provide file and print sharing services to users via the network, are the first point of penetration for many office networks because operating systems (OSes) provided as open source are capable of providing the same services as proprietary operating systems. For instance, there are open source implementations of the UNIX Network File System, Microsoft Windows file-sharing, Apple AppleTalk file-sharing, and NetWare file and print services, among others.

In this way, open source OS servers can become a point of entry into companies. Whether servers run a proprietary OS or an open source OS is immaterial to corporate users, since they interact with the servers exclusively via the network, and open source OSes provide the same network services. In this way, many small- to medium-size companies ease into open source software on their servers.[1] To the company, it is simply receiving a better deal, as its consultants can support the open source OS server just as well or better than they could support a proprietary OS server.

4.2 Legacy applications

Years ago, my family bought a hand scanner that came with MS-DOS driver software. When Windows 3.0 was released, its manufacturer discontinued application support for it. Later, we bought a new scanner from the same manufacturer. However, when Windows 95 was released, the manufacturer discontinued support for this newer scanner.

[1] This is not to say that large companies are not also beginning to use open source software. However, large companies generally have an internal I.T. department rather than hiring outside consultants.

If the manufacturer in the above scenario had made specifications available for its hardware, then these scanners could still have been supported by third parties. But since specifications were not available, the hardware could no longer be used once the original manufacturer abandoned it. Similar issues arise often, even when hardware is not involved.

Corporate environments often suffer the problems of supporting legacy applications. A "legacy app" is a program that has been abandoned by its original author, leaving its users in the lurch. When a company has based its business on that program for the last several years, it may mean that the company must continue to support it for a number of years further, until a suitable replacement can be found or developed. As other programs continue to change, and the legacy app stays in one place, it causes more and more difficult support problems. A single legacy app can impede an entire company's progress.

Open source software cannot create legacy apps. Although a program can be abandoned by its creator, it can then be adopted by another person or company. Instead of forcing companies that depend on an abandoned program to avoid upgrading other software, this allows them to continue development of the program and fix incompatibilities with new versions of other software.

4.3 User friendliness

Corporate and home user machines are an issue separate from servers. Since users interact with servers only through the network, it is easy to change the OS used on the servers. The users' own machines are a separate issue. Features found in open source software, such as high reliability and high performance, may be less important than the users' ability to easily use the machines. It cannot be assumed that every user is a computer expert.

Unfortunately, there is not an exact open source equivalent for every common office application. Traditionally, open source software has been written by programmers for programmers. This leads to small, powerful programs that are not necessarily perceived by neophytes as user friendly. For instance, TeX is a popular open source word processor. It is powerful and flexible, but it requires users to learn a sometimes-complex language to use its advanced features. Users who are accustomed to Windows-based word processors will encounter challenges using TeX.

Open source applications designed for new users are on the way. By the end of 1998, it is likely that there will be user friendly open source equivalents for all common office applications.

4.4 Technical support options

For those who decide to take the plunge, there are several technical support options. First of all, traditional fee-based commercial support is available for complete open source operating systems (e.g., Red Hat, Caldera, Debian) as well as individual programs (e.g., Samba, GIMP, Sendmail). This support is provided in a model similar to support for proprietary software. Paid commercial support is ideal for users who need assistance on a regular basis.

But open source software offers forms of support not usually available for proprietary products. Let's consider them in the order that they would be tried by a typical user searching for information on how to use a particular product.

Web pages are the first valuable resource. Most open source programs have a web page dedicated to them. These web pages include a copy of the program's documentation, but in addition they often have a list of Frequently Asked Questions (FAQs) with answers. The most common questions about a program can be answered by referring to the FAQ list. Some programs also have an associated HOWTO document, which gives step by step instructions for installation and use.

Newsgroups and email lists are the next step. These are both forums where users can interact to ask and answer questions. Most programs have at least one email list of this sort associated with them. It is common to receive an answer within ten minutes of the question. Often, when a question is asked more than a few times, the question and answer is added to the program's FAQ list.

Internet Relay Chat (IRC) is another popular resource. IRC is a "chat room" protocol that allows users to ask questions and receive answers in real time. It is invaluable for giving interactive assistance since each step can be performed as it is described.

The quality of this multilevel support cannot be overestimated. Users of the open source operating system GNU/Linux, for instance, can easily find answers to their questions. In fact, support for Linux is so good that InfoWorld gave its 1997 Product of the Year for Best Technical Support to the Linux user community.

4.5 Social impact

Companies frequently cite lack of technical support as a reason for avoiding the use of open source software. We have just noted, however, that due to the availability of commercial as well as cooperative technical support, support for open source programs is better than support for proprietary closed source software. Companies should regard the support framework for open source software as an asset rather than a liability.

The social structure of open source software encourages the formation of communities consisting of both users and programmers. This allows better support than with partitioned groups of either programmers or users by themselves. Programmers can help users to differentiate between user error and genuine bugs, and first-hand accounts from users can help programmers to plan usability and feature improvements for future versions.

Proprietary closed source software, with the intermediation between programmers and users that implies (see Section 2.3 [Social Impact (Environment)], page 4), does not allow for the formation of communities of users and programmers together. Programmers are isolated within the software company, seeing only what the marketing and public relations department chooses to tell them. Users are isolated outside the company, without direct access to those most knowledgeable about the software, its programmers.

5 An open future

When a proprietary software company builds a new closed source program, the effort is driven by marketing. Hype and advertising are paramount, the software itself secondary. Compatibility and prevention of defects is less important than driving revenues.

This paper has tried to demonstrate why the proprietary closed source software development model is economically inefficient compared to the open source software model. The above sections make an effective economic case against closed source software. Let us now look beyond economic rationales, to social aspects of open source software.

Years ago, there was no market for proprietary software. At the dawn of the digital era, when a single computer occupied entire rooms, software was not a commodity to be sold. Instead, it was shared, spread around the computer room, used, copied, examined, and improved by anyone interested. An open source community existed.

As the years passed, and computers shrunk in size and increased in number, the community all but disappeared. There was no feeling of camaraderie among the microcomputer programmers from the late '70s through the '80s. There was no forum for them to interact and trade information.

With the rise of the Internet in the '90s, open source software has again brought together a community. The Internet provides a place where thousands can gather to build open source software for the present and the future. Writing software is again a community activity.

The motivations of those in this new open source community are to write quality software for themselves. Studies have found that those who perform creative work due to intrinsic motivations do better work than those who are promised rewards for performance. This applies to professional programmers as easily as to the studies' subject groups. Open source programmers write what they do because they want to, so they produce better work.

Open source programmers in the community take pride in exhibiting their work to the outside world. It's the same reason that classic car hobbyists take their cars to shows: because they want to show off their talents. Open source programmers polish their software and make it shine.

Open source software is growing in popularity, and the rate of growth is increasing. Publicity is getting to be more and more effective. More and more proprietary software companies are recognizing the value of open source. For instance, Netscape released Mozilla, an open source version of Communicator, to the public in early 1998. Even Microsoft is hinting that it may make some of Windows NT's source code available to the public.[1]

Open source is on the way up. This is good for everyone. Here's to a bright open source future: Share and Enjoy!

[1] This may not mean that the source code will be released under an open source license, but it is a good sign.

6 Bibliography

Free Software Foundation [1997]. "Free Software is More Reliable!" (http://www.gnu.org/software/reliability.html).

Ishee, David [1998]. "Fear of X.0" (http://slashdot.org/features/9806151013252.shtml).

Kohn, Alfie [1987]. "Studies Find Reward Often No Motivator." *Boston Globe*, 19 January 1987. Available online at http://www.gnu.org/philosophy/motivation.html.

Raymond, Eric [1997a]. "The Cathedral and the Bazaar" (http://earthspace.net/~esr/writings/cathedral-bazaar/).

Raymond, Eric [1997b]. "Homesteading the Noosphere" (http://earthspace.net/~esr/writings/homesteading/).

Raymond, Eric [1998]. "The Open Source Page" (http://www.opensource.org).

Schulman, Andrew [1992]. *Undocumented Windows*. Addison-Wesley, Reading, Mass.

Stallman, Richard [1985]. "The GNU Manifesto" (http://www.gnu.org/gnu/manifesto.html).

Stallman, Richard [1994]. "Why Software Should Not Have Owners" (http://www.gnu.org/philosophy/why-free.html).

Stallman, Richard [1997]. "Why 'Free Software' is better than 'Open Source' " (http://www.gnu.org/philosophy/free-software-for-freedom.html).

Table of Contents