The Ethics of Free Software

John Goerzen, jgoerzen@complete.org

December, 1998 (revision 2)

Available online in PostScript and hypertext at http://www.complete.org/papers/fsethics/

Part I

Introduction

1 The Failure of Proprietary Software

"Using Windows NT [closed software], which is known to have . . . failure modes, on a warship is similar to hoping that luck will be in our favor." – Anthony DiGorgio, Engineer, United States Atlantic Fleet Technical Support Center [19]

It was a fairly normal day for the USS Yorktown in September, 1997. The Aegis Missile Cruiser was participating in maneuvers off the coast of Cape Charles, Virginia. No enemy ships were in sight.

Suddenly, the ship's entire propulsion system inexplicably failed. The USS Yorktown was dead in the water, but the engines were completely normal. [19]

What, then, caused the problem that required the 80,000 horsepower ship to be towed into port for over 48 hours of maintenance?

It turns out that the difficulty encountered there was but one symptom of a serious problem affecting the very methods used to write the code responsible for running nine out of every ten of the world's microcomputers. The USS Yorktown failure is only one example of what can happen when trust is placed in inherently untrustworthy systems and design methodologies. Systems can crash on a daily or even hourly basis, [6, #reliability] with data loss a potential unfortunate reality when such events occur. [1] Critical systems can stop functioning, costing millions of dollars in lost revenue. In medical and space fields, for instance, lives could be lost.

The problem behind all this is *proprietary software*. Proprietary (or closed) software lacks many of the benefits that society has derived from the marvels of the industrial resolution. When a proprietary project is developed, there is no peer review. Imagine taking a flight on a jumbo jet designed by only a single person with no safety review from others. The thought rightly sends shivers up your spine. You're of course aware that humans are imperfect and mistakes do happen. Peer review is a key way to find the mistakes *before* they have serious consequences, such as an airplane crash, a bridge collapse, or the crippling of a warship.

Closed software also deprives computer scientists and developers from a chance to learn from those that have gone before. Imagine if Newton, for instance, didn't share his discoveries with others. Mathematics, physics, and scientific methods would certainly have taken a much longer time to develop, since many things in those fields have built upon Newton's ideas. [22]

Even though we have a long track record of the success of building upon previous innovations in virtually every field of science, individuals and organizations creating computer software and hardware conspicuously take the opposite view. Not only is this simply harmful to the quality and quantity of software and hardware available, it also is not the ethical thing to do.

2 The Free Software Solution

As we've seen above, there are tremendous problems with using and developing closed software. Fortunately, there is a solution: Free software (also known as Open Source[1]). When we talk about free software, we refer to freedom, not price; it's possible to charge for free software. Free software solves many of these problems by ensuring that not only can we build upon previous good software, but we also have an opportunity to have peer review of current software. This is often accomplished by licensing software under the GNU General Public License (GPL) [20] or a similar license.

Part II

A Utilitarian Analysis of Free Software

The utilitarian principle can be defined as:

> Everyone ought to act so as to bring about the greatest amount of happiness for the greatest number of people. [5, page 24]

Modern interpretations of this principle often can focus on things that lead to fulfillment and a rewarding life. In this context, the right thing to do, ethically, is that which maximizes the total benefit ("utility") to people. In order to show that free software is ethically the correct choice, it must be shown that, when compared to alternative development paradigms or licensing schemes, free software is the most beneficial for the greatest number of people.

[1]The meaning of the term Free Software is equivolent with that of Open Source Software. [13]

3 Software Quality & Reliability

Software has a tremendously broad reach in today's society. Almost every person in the United States is affected by software, either directly or indirectly. Everything from surface mail and phone calls to airline flights and Internet-based commerce is handled by software. When the software behind these various activities is of higher quality, people that use this software benefit because it performs better and is more reliable.

They are happier because their airline flights are on time. This means people are less likely to miss an important meeting or a connecting flight at their destination. Internet orders are more likely to arrive correctly, meaning that the birthday gift for someone would be timely. To put it another way, poor software quality causes a great deal of harm. Errors in billing, credit reports, tax records, and the like are but a few examples of things attributed to software glitches. Any of these things can cause significant harm to people or companies. People can be unfairly targeted by collection agencies. They may waste money defending themselves against something they didn't do. Time is lost dealing with those people. A rejected loan, for example, could mean that a growing family can't get a larger home that they need. All of this can lead from low-quality software – and it does happen.

Software developers also benefit from better design methods. They can produce software in a shorter amount of time, and the computerized tools that assist them in development make their job easier.

Therefore, if it can be shown that free software produces higher-quality software, then it follows that we achieve greater productivity and more benefit for mankind, and free software is the ethical choice for software developers. Below, several attributes of free software are discussed, along with the reasons that they help to improve software quality.

3.1 Peer Review

One of the most important aspects in the design of any large and important project is peer review. In large projects such as operating systems, hundreds or even thousands of people work on that which ends up being millions of lines of code. A single missing character in that code can be enough to allow a security breach or cause a system crash. Humans are not perfect, and while computer scanning programs can catch some simple mistakes, there are many mistakes that can only be caught by another person looking at the code.

With closed software, in general, only employees of the company writing the program have access to its source code. This makes it impossible for others outside the company to find problems in the code *before* they cause damage. Worse, when there is a problem, the users of a program can do nothing to help find it.

With free software, such as Linux, the source is freely available for download. People are encouraged to look at it, to be critical, and to try to find bugs. And thousands of people do look at the code. The end result is that there are far more people proofreading and fixing code, and the program has fewer bugs and is more stable. [8]

3.2 Timely Fixes

When no source code is available, many bad things can happen, from security problems to devastating system crashes. The USS Yorktown is a great example. Do we really want to prevent the military, airlines, or any other industry in which lives could be at stake from being able to fix system problems in the field? If something breaks, they are denied the ability to fix it because they have no source code for the software! This isn't the only situation where vital government systems have been hampered by closed software. On December 15, 1998, a bug in Microsoft's closed mail server called Exchange took down two critical servers in the United States House of Representatives. Ironically, the bug occurred just when e-mail traffic was at its peak – days before a critical vote on the future of the President. House members, therefore, couldn't receive valuable feedback from their constituents. [4]

Both these situations were caused by bugs in proprietary code. In both situations, the people running the computers were completely helpless. They could not have fixed the problem regardless of how much effort they put into it; the lack of source code completely prevents that. A computer glitch is bad enough, but when people are denied the ability to fix it themselves, or to hire someone else to fix it in a timely manner, there can be very serious consequences. Lives aboard crippled ships could be lost. Vital government business could grind to a halt as the communications lines between the people and their representatives are severed.

Free software promises a better world. In the free software world, anyone can fix a problem with the software in use. Even if a company does not have the expertise in-house to fix the problem, contractors are plentiful, and they can make quick fixes. In situations where even an hour of downtime can mean literally millions of dollars in lost revenue, having a fix in a matter of hours or even minutes as opposed to days or weeks can make a tremendous difference.

3.3 Users Are Developers

This is a concept completely foreign to the closed source world, but it alone is of tremendous importance in the free software community. This concept is important in two separate areas: debugging and development.

With proprietary software, when a bug is found, even if it is serious, it often takes a long time for it to be fixed, if it ever is. A lot of things can go wrong. If the original manufacturer of the software program no longer exists, and the source to the program is not available, it is generally *impossible* to fix the problem. If this program is essential to someone's business and no longer works, the company has a big problem. This is causing tremendous difficulty as some software will break when the year 2000 arrives, but people using it have no way to fix the problem! As Raymond puts it, reliance on close-source software in the face of the year 2000 problem could "kill your business." [12]

This is only a small part of the problem. If your software vendor is unable or unwilling to fix problems, you are stuck with broken software, and there's nothing that can be done about it.

For software vital to the operation of businesses, such as billing and financial software, these consequences are plenty to demonstrate the ethical problems with keeping the source code to software a secret.

Free software presents a better alternative. When you find a bug in your software, you can fix it yourself instead of depending on the original vendor to fix it. Or, somebody else can be hired to fix the problem. This instantly takes a difficult, possibly insurmountable problem with software, down to something that could be fixed in a matter of hours or even minutes in many cases – a clear benefit of free software.

Another important aspect of this is that the users of a given free program can add their own features to it. If you're not satisfied with how something works, you can improve it! This capability is especially useful if the software vendor does not exist any longer or is unwilling to implement your desired changes. You get a better product, and the people using it are more productive.

The benefits are even greater than that, though. When people find bugs or add new features on their own to free software, they are encouraged to submit their changes back to the maintainers of the software. This means that every user of an free program is a potential contributor to it! What's more, if people have made modifications to a program – new features or changes to fit their taste – there is free peer review of these modifications! When people that add features cooperate with the process, which is almost always the case, the other people using the software can see the code and can spot problems. As a result, free software products can have bugs found and fixed faster, new features implemented sooner, and better reliability than closed-source products. [11] People that make changes for their own use can get these changes peer-reviewed, increasing the quality of their software. The result is that software is less likely to crash, systems are down less, and the software better meets the needs of those that use it – all of which contributes to an increase in utility.

3.4 Security

Security is one of the most complex areas of software development, requiring expert programmers to write secure code and find security problems in existing code. With closed software, the number of people that are able to review the security of code is limited to a miniscule fraction of the total programmers that could do this.

There are many instances of security problems in closed-source software that never existed or have long ago been fixed in free software. These problems often can cause serious loss of important data, which can easily lead to devastating consequences. For instance, on July 8, 1997, the United States Coast Guard's database server crashed and remained down for 1,800 hours while 115 employees worked to restore the data. [3] In the first week of March 1998, attacks caused *thousands* of Windows NT systems to crash, exploiting a security hole in that proprietary operating system, including some particular sites that suffered over one hundred failures. [17] These are just a few examples of exploitation of closed software bugs by crackers[2] – bugs that are not present in free software.

The cause, however, goes deeper than the general issues of free software discussed above. In a concept called "security through obscurity," people try to keep their data secure or encrypted not by necessarily writing secure software, but rather by tightly restricting access to the source. The theory behind this is that if those attempting to crack systems do not know how the algorithms work, then they will not be able to breach security.

This argument fails quite easily, however. In [10], Mr. Perens states, "Security through obscurity is always a bad idea." While that statement is perhaps too bold, its sense is correct, and the reasoning behind it can be analyzed in terms of utilitarian ethics.

Many popular security systems and encryption algorithms are presently compromised, but manufacturers of such software continue selling it. This is often because others cannot review the code, and the manufacturers themselves may not even possess the knowledge necessary to find or fix the problems.

[2] Often incorrectly called "hackers" by the media; the correct term is "cracker". [15]

Software developers can conceal security holes in their software, [10] intentionally or unintentionally. The end result is that people think that their data is secure when it really may not be. The consequences of this can result in loss of utility. A simple hypothetical will suffice: if the data on a courtroom computer holding privileged information about people's criminal records is leaked, people's careers and lives could be destroyed based on that information. Other cases of broken security could have a serious effect on national security, even causing loss of life in some situations.

When we have free software algorithms, we can be absolutely certain about their security level. This places the power to decide whether a given algorithm is appropriate in the hands of the users of it, not its author. The users of such algorithms can have a high degree of confidence that it is sufficient for their needs, thus increasing utility by decreasing not only the chances of compromise, but also extra efforts and worry needed to deal with it.

4 Legal and Monetary Issues

There are many different licenses that qualify as free software according to the Open Source Definition [2]. These include the BSD[3] license, the Artistic[4] license, the MIT/X Consortium[5] license, Netscape's Mozilla Public License[6], and the Free Software Foundation's GNU General Public License[7] [20]. Of these, I shall be discussing the GNU General Public License (GPL) in greatest detail. It is the most popular way to license software in the free software community, and is considered by many to be the best license under which to place software in order to ensure that it remains free forever.

In order to show that legal issues matter in a strict utilitarian evaluation, it has to be shown that these issues affect utility. With legal issues, this relationship can come in two forms: 1)

[3]http://www.opensource.org/bsd-license.html
[4]http://www.opensource.org/artistic-license.html
[5]http://www.opensource.org/mit-license.html
[6]http://www.mozilla.org/NPL/MPL-1.0.html
[7]http://www.fsf.org/copyleft/gpl.html

saving money on software and legal expenses and 2) saving time relating to legal issues.

If money can be saved, there is a benefit of free software. If a company is being discussed, the company's operation becomes more efficient. They can either spend the money they have saved on other things such as research, or pass the savings on to their customers. For individuals, the money can be used for other things – paying bills, household expenses, paying off a mortgage, etc.

Saving time also leads to benefits. This issue applies primarily to companies. If they save time dealing with legal issues, they become more efficient, leading to the same consequences as saving money. They can put more resources into development, for instance, which benefits not only themselves but everyone that uses their products.

4.1 The GNU General Public License

The GPL is the license under which most free software is placed. The idea of the GPL is to ensure that a given piece of software will always remain free, and that others can use part or all of it in their free software projects, subject to a few restrictions. In a nutshell, this is accomplished by requiring that sources be distributed alongside any compiled version of the program, and requiring that any program that uses some GPL code must itself be under the GPL. The GPL only restricts the ability to copy the program in that if it is given to others, the sources must be made available, too. People can use any GPL program on as many of their machines as they wish. GPL does not address cost; it is possible to charge for GPL software, but the cost is often very low or non-existant since it is legal to make a free copy of someone else's software.

4.2 Businesses Benefit from the GPL

In today's world, most companies pay a large amount of money for software. With GPL software, people are allowed to make as many copies as they want, for no extra charge. Most GPL software can be downloaded off the Internet at no cost. A few programs have to be purchased by CD-ROM or some other non-Internet method, but once the program had been obtained, it is legal (and encouraged) to use it on as many computers as desired. For instance, if Compaq suddenly decided to pre-install Linux on the computers they sell instead of Windows, they would go from paying Microsoft $750 million per year [16, page 2] to paying a Linux CD vendor $7. While this is an extreme example, the general idea is there: businesses spend a lot of money for software.

There are indirect costs as well. Businesses must keep meticulous record of the software that they have, and the associated licenses, to make sure that they do not violate any licenses. If these records are not kept, or contain a single mistake, there is a risk that the business could face an expensive lawsuit. Keeping records takes time, and even if records are kept, there is still a possibility of a lawsuit.

Businesses must divert more time and energy from the things important to their company to meticulous record-keeping or perhaps even liability insurance. This means that either they must charge their customer more or they have fewer resources to devote to support, research, and development. This can lead to harm outside the business as well.

4.3 Individuals Benefit from the GPL

People have budgets, too. Sometimes, people are forced to avoid using a particular software program simply because they cannot afford it. This is certainly not beneficial. If everyone could afford the software that they want or need, they'd be better off because their desires are fulfilled, and they'd most likely be more productive, since they have software that fits their needs better.

An example of this is tax preparation software. Consider the situation of somebody that cannot afford to have an accountant prepare his taxes, and cannot afford to purchase a software program to help him. This person could very well make mistakes on his tax return, which could result in serious fines from the government, making his financial situation even more precarious. Obviously, there is a net loss of utility with this sort of proprietary software.

Now consider if he were to use a GPL tax-preparation program. Not only would the program itself likely be of higher quality for reasons already outlined in section 3, but also he would likely be able to download the program and use it at no cost to himself. His tax return will more likely be correct, and he has a smaller chance of getting a nasty fine from the government.

5 Educational Value

In order to be a good programmer for something as complex as an operating system, years of experience with programming are generally required. One can learn the basic concepts behind programming, but this doesn't give much "real-world" experience. One tremendous benefit of free software is that every free software program instantly becomes a valuable educational tool.

One great way for programmers to learn is to look at good code from others. This is precisely the sort of advantage that free software provides. Programming students can study or even modify the internal workings of the Linux operating system as class projects, for instance. Hands-on experience with a modern operating system is a great way to learn skills.

If people are better-educated, we get higher-quality software, which of course leads to increased utility – a definite win.

Another benefit here is that students can use, at no cost, a free software operating system such as Debian GNU/Linux or FreeBSD at home. This gives them the same computing environment as they get at their place of learning, with the added benefit that they can tinker with the source code to every part of the system to their heart's content.

5.1 Learning Becomes Fun

One current problem facing the United States is a shortage of skilled technical workers and programmers. [23] A great way to combat this problem is to get more people interested in computer science, electrical engineering, and related fields of study. As Stallman points out in [21, Why People Will Develop Software], programming is fun. If somebody enjoys programming, it's a great way to get them into the field. The free software concept offers ideal ways to not only get started with programming, but incentives to keep going. There's a certain feeling of satisfaction for a programmer when he makes his first patch to the Linux kernel or fixes a first bug in an e-mail client. Having the source to tinker with is an excellent opportunity to recruit people.

6 Possible Objections

These arguments may not be enough to convince some people. There are frequently arguments advanced in opposition to the free software concept. Raymond refutes some of them in [13], but he doesn't take a utilitarian or even an ethical perspective in his arguments. Here, some of these objections are phrased in utilitarian terms, with corresponding answers.

6.1 Programmers Need Pay

This is often the first objection to the free software idea. The general idea is, "if software is available at no cost, then nobody will be able to pay the programmers, and they will have to give up their coding skills and find another profession." In utilitarian terms, the argument means "since programmers can't get paid for no-cost software, they will either starve or have to give up their chosen profession, causing unhappiness for them."

However, this argument doesn't work because the premise is flawed for several reasons. First, current estimates put the amount of code written for companies to be used in-house at over 75%. [14] The people to write this code will have to be hired in any case. Then, there is the fact that most time spent in software development is maintenance – fixing bugs, adding new features, and changing the program to meet new demands.

6.2 There's No Liability

A lot of corporate officials want to have a "cushion" in case something goes wrong with their software. That is, if the software doesn't work, they want to be able to sue the vendor. To put this argument in utilitarian terms, if a company is hurt because of the problems in the software sold by somebody else, the hurt company wants payment to make up for that harm.

However, this sort of argument ignores several issues. First, much of the most important or most common software for the computer, such as the operating system, is sold under a license that explicitly releases the software vendor from any such responsibility. In many cases, the vendor doesn't even guarantee that it's useful for what it's sold for. It would be difficult or impossible to collect any sort of money from such a vendor.

Secondly, this argument also ignores the fact that free software is less likely to have problems in the first place. If the software is less likely to have problems, there's a smaller chance that there is going to be any harm at all – a utilitarian win.

Third, there are various support companies that can provide guaranteed response time to problems with free software.

Finally, the argument ignores the legal time and cost involved with suing a software vendor. Sometimes this expense is so prohibitive that the hurt company will just take the loss and not try to sue anybody.

6.3 National Security

In some cases, releasing source code to the general public could be harmful. These cases are extremely rare, and can generally be considered to be limited to that code which is responsible for controlling systems that have significant physical destructive power. This is similar to existing precedent. For instance, while it is generally agreed that exchange of knowledge and ideas is good, the consequences of widespread dissemination of information detailing how to build an atomic warhead override the normal considerations. This is not a strike against free software; rather, it's a reflection of the unfortunate reality that software is used to kill, and some knowledge ensures human safety better by withholding it. Source code, in this case, simply acts as something that carries the knowledge of these devices. One could write down the information on paper and achieve the same effect. The problem lies not with free software, but rather with the awful burden software is being asked to carry. In utilitarian terms, we can clearly see a great possible harm if knowledge about anything that can cause significant physical destruction gets into the wrong hands.

6.4 Programming Competency

Many of the benefits derived from free software come about due to the efforts of programmers around the world. For instance, a company might realize that it's great to be able to fix problems when they occur, but the company may not have anyone on staff capable of fixing programming mistakes or making custom enhancements.

Fortunately, this is not really a problem. First, if good problem reports are sent to the author, a task which does not require programming skill, most free software authors respond to them in an amazingly small amount of time. Secondly, consultants or contract programmers that understand free software are plentiful. Finding solutions to problems does not require programming knowledge; there are numerous ways to get support for free software programs.

7 Free Software Successes

The ideas laid out above show why the free software idea makes sense ethically. The free software community has taken advantage of its unique development paradigm to create many truly amazing pieces of software. Places like Metalab[8] are huge repositories of free software, containing literally thousands of free software packages with full source code. Free software has been a major player in the Internet, and hence the global communications revolution that it has caused. The benefits from the contributions of free software to society and global communication are tremendous. While it is not possible to provide an in-depth analysis of each of thousands of packages and the good they've done for computing and the world, it is at least possible to analyze the improvements that some of them have brought to us because of the free software paradigm.

[8]http://metalab.unc.edu/

7.1 Apache

The Apache[9] free-software web browser is currently the world's most popular. 53% of all Internet world-wide web servers run Apache, more than twice that of all the Microsoft servers combined, its nearest competitor. [9] Due to its open development model, Apache has attracted new features faster than anyone else. Programmers on the Internet contribute to the product and make it better.

Apache is dominant for several reasons. One is that it is arguably the most featureful web server in existence today. This stems from the fact that a free-software project such as Apache can tap a huge resource of programmers to contribute to the code and make it better. Another is that Apache is free – both in terms of cost and in terms of freedom to use it as desired. Small companies, home computing users, poorer countries, charities, under-funded organizations, and the like can all afford to run Apache. Few, if any, of them could afford to run the commercial servers. Because Apache is available a no cost, many people are able to publish their ideas and information that otherwise would go unpublished. Apache provides stability that is unmatched by its commercial competitors, thanks in large part to the free software peer review process.

7.2 Sendmail

For many years, the single largest consumer of traffic on the Internet was e-mail. It still uses a large amount of bandwidth worldwide, and is working to revolutionize our communications. A free software program has been largely responsible for this. That program is sendmail[10]. Today, sendmail powers over 75% of the world's Internet e-mail servers. [18] Like Apache, sendmail has done its part to usher in a communications revolution. Because it costs nothing to use, anyone can have a mail server. Sendmail is widely regarded as being the most powerful mail server available today, and this is mostly because of the free software design paradigm.

As living proof that programmers working on free software can make money, one need look

[9]http://www.apache.org/
[10]http://www.sendmail.org/

no farther than sendmail. Its authors have set up a profitable business that sells commercial support contracts and related items for sendmail.

7.3 Mozilla

On March 31, 1998, Netscape Communications Corp. did something that shook the industry: they released the source code to their next web browser, code-named Mozilla, as free software. Never before had such a large and well-known closed-software program been converted to the free software paradigm. Netscape, the world's most popular browser, had been losing market share to Microsoft's browser. Netscape needed a way to pull ahead, both in terms of features and development speed. Free software provided that for them.

Since that time, Mozilla[11] has taken off. It now has features to make the Internet more accessible to people. Mozilla uses less memory than other browsers, and yet can do more. It is conceivable that Mozilla will be small enough to provide full-featured Internet access on palm-sized computers in the near future. Mozilla has features to speed web browsing by 30%. Developers have ported Mozilla to a huge number of different types and architectures of computers even though Mozilla has yet to be released as a finished product. There are already many people that can use the Web that could not before.

7.4 Linux

Apache, sendmail, and Mozilla are all important. Apache and sendmail have played an important part in revolutionizing Internet communications. From the looks of things, Mozilla will do so some day as well. All of this is great, but there's another free-software phenomenon that's taken the closed-source community by surprise: Linux. Linux is an operating system written entirely using free software. Since development started in 1991, Linux has grown at a tremendous rate. Linux sports very rapid development speed, excellent performance and stability. The Linux operating system is used for both running and developing other free software projects as have

[11] http://www.mozilla.org/

been described here. In many ways, Linux is better than closed operating systems. Linux's stability is legendary, as is its speed. Linux has done a lot to help spread the word of the free software revolution. Programming students now have a stable platform to use for development. Businesses don't have to worry about crashes. People with slower computers don't have to worry about upgrading, in many cases.

In the eyes of some Linux advocates, however, Linux's greatest achievement has yet to be finished, but it is well on its way. Due to the extreme importance of the operating system on any modern computer, Linux is in a great position to demonstrate to the world the benefits of free software. Many feel that Linux could one day topple the closed-software paradigm and the huge companies that encourage it. Already, evidence of this is emerging. Forbes reports that RedHat Software, for instance, has projected revenues of $10 million in 1998. [7] RedHat is almost exclusively selling free Linux software.

As we've seen above, the free software paradigm has many advantages. Linux is helping to bring the free software paradigm to the masses, and as such, is doing a tremendous favor for computing.

8 Conclusions

Given all of these ethical arguments and the tremendous amount of good that we've seen free software has made possible in our world, it's easy to see how free software makes sense from a utilitarian perspective. Most of the advantages of free software end up being in higher-quality software, lower cost, and smaller development time. All of these lead to tremendous benefits for mankind, and so I conclude that free software is a better paradigm, ethically, than proprietary software.

References

[1] Mike Andrews. Massive NT outage due to registry corruption. *ACM Committee on Computers and Public Policy: Forum on Risks to the Public in Computers and Related Systems*, 19(60), February 1998. http://catless.ncl.ac.uk/Risks/19.60.html#subj10.

[2] Debian GNU/Linux Developers and Bruce Perens. *The Open Source Definition*. Software in the Public Interest, Inc., 1998. http://www.opensource.org/osd.html.

[3] Laura DiDio. U.S. Coast Guard beefs up security after hack. *CNN/Computerworld*, July 22, 1998. http://cnn.com/TECH/computing/9807/22/coastguard.idg/index.html.

[4] Sandra Gittlen and Jason Meserve. Bug slows impeachment e-mail to House members. *CNN/NetworkWorld*, December 16, 1998. http://www.cnn.com/TECH/computing/9812/16/housemail.idg/index.html.

[5] Deborah G. Johnson. *Computer Ethics*. Prentice Hall, second edition, 1994.

[6] John Kirch. Microsoft Windows NT Server 4.0 versus UNIX, November 1998. http://www.unix-vs-nt.org/kirch/.

[7] Josh McHugh. Freeware children: For the love of hacking. *Forbes Magazine*, August 10, 1998. http://www.forbes.com/forbes/98/0810/6203094a.htm.

[8] Metro Detroit Linux Users Group. Linux presentation online: Advantages of linux: Code is subject to peer review, October 1998. http://www.tir.com/%7Esorceror/mdlug/2-1/review.html.

[9] Netcraft Ltd. Netcraft web server survey, December 1998. http://www.netcraft.com/Survey/.

[10] Bruce Perens. Why security-through-obscurity doesn't work. *Slashdot*, July 20, 1998. http://slashdot.org/features/980720/0819202.shtml.

[11] Eric S. Raymond. The cathedral and the bazaar. Revision 1.40, http://www.tuxedo.org/˜esr/writings/cathedral-bazaar/.

[12] Eric S. Raymond. Closed source after 2000? http://www.opensource.org/y2k.html.

[13] Eric S. Raymond. Frequently asked questions about open source. http://www.opensource.org/faq.html.

[14] Eric S. Raymond. Yes, you can eat open source. http://www.opensource.org/open-jobs.html.

[15] Eric S. Raymond, editor. *The New Hacker's Dictionary.* MIT Press, third edition, September 1996. Available online at http://www.tuxedo.org/jargon/.

[16] Reuters. Dues due for windows? *Wired News*, November 20, 1998. http://www.wired.com/news/news/politics/story/16386.html.

[17] Reuters. Hacker attacks target Windows NT computers. *CNN*, March 4, 1998. http://cnn.com/TECH/computing/9803/04/microsoft.attack/index.html.

[18] Sendmail, Inc. Sendmail products. http://www.sendmail.com/products/smpro.html.

[19] Gregory Slabodkin. Software glitches leave Navy smart ship dead in the water. *Government Computing News*, July 13, 1998. http://www.gcn.com/gcn/1998/July13/cov2.htm.

[20] Richard M. Stallman. *GNU General Public License.* Free Software Foundation, second edition, June 1991. http://www.fsf.org/copyleft/gpl.html.

[21] Richard M. Stallman. Why software should be free, November 1, 1998. http://www.fsf.org/philosophy/shouldbefree.html.

[22] Eric Weisstein. Treasure trove of scientific biography: Newton. http://www.astro.virginia.edu/˜eww6n/bios/Newton.html, 1998.

[23] Brad Wieners. Closing the geek gap. *Wired News*, January 12, 1998. http://www.wired.com/news/news/technology/story/9624.html.