Why Not Use the GPL?
Thoughts on Free and Open-Source Software

Doug Palmer*

February 15, 2003

Contents

1 Introduction

I make a living developing commercial software for a company that fully expects other companies to pay for the use of that software.[32] I also develop and distribute snippets of open-source software, although less than I would ideally like to. Over the years, I've made full use of the tools offered by the Free Software Foundation (FSF) under the terms of the *GNU General Public License* (GPL).[11, 7] In fact, `gcc` was the only compiler that could successfully compile the enormous switch statement at the heart of my PhD research. I'm an enthusiastic user of the GPL'd Linux operating system. Most of the programs that I use are open-source in some way or another. This document was typeset using LaTeX.[33]

I am now about to bite the hand that feeds me . . .

2 Morality

The FSF takes a strong moral position with respect to "free software". Those who sell proprietary software are denounced in fairly uncompromising terms.[10, 16, 27] Unfortunately, "free" is a word that tends to be subject to creative re-definition.[19]

*mailto:doug@charvolant.org http://www.charvolant.org/~doug

2.1 Free as in Speech?

> Free software is a matter of liberty, not price. To understand the concept, you should think of "free speech", not "free beer".[9]

The article from which the above quote was drawn lists four basic freedoms underlying the definition of free software. The freedoms are:

0 The freedom to run the program, for any purpose.[1]

1 The freedom to study how the program works, and adapt it to your needs.

2 The freedom to redistribute copies so you can help your neighbor.

3 The freedom to improve the program, and release your improvements to the public, so that the whole community benefits.

These freedoms talk in terms of a *program*. However, the general terminology is *free software* and I'm going to take it for granted that we're talking about programs, libraries, graphics, message catalogs, help files and all kinds of other software artifacts.

If free speech is taken to mean the right to express an idea then free software has very little to do with free speech. In software terms, *free speech* means the right to write a program or library that performs certain tasks. Free software does nothing to protect the right to write a program about something, although most subscribers to the free software ideal would, presumably, support such a right.

Freedoms 1 and 3 imply access to source code: open-source. With the exception of access to source code, the four freedoms simply amount to what is normally termed the *public domain*. And, since placing source code in the public domain is well within the generally accepted meaning of the term, access to source code is not a particularly strong extension to the concept.

If free software is little more than coupling the notion of open-source to the public domain, why surround free software with special licenses and manifestoes? I think the answer lies in the free source movement being a political movement. And, like most political movements, the aim is not to allow people to do as they will, but to force people to do as the movement decrees.[2] This is a pretty strong statement, but I think the form of the GPL itself (discussed in section 2.2) and articles at the FSF's web site support this position.

As well as the GPL, the FSF provides a *Lesser GPL* (LGPL).[8] For various technical reasons, most of the software I write is under the terms of the LGPL. When examining the text of the LGPL at the FSF site, I was surprised to encounter a link to an article advocating placing libraries under the full GPL.[30] The main thrust of the article seems to be that, by placing libraries under the GPL, users of the libraries are forced to place the programs that use the libraries under the GPL, as well. This article is *explicitly advocating restrictions on the use of software*.

2.2 What's Wrong With the GPL?

A Warning: I am not a lawyer. Nor do I play one on TV. The following is a programmer trying to get to grips with a licensing document, not a legal opinion. It's entirely possible that I've missed some "obvious" piece of law that directly effects my interpretation.

Examining the GPL, there are a number of knotty clauses:

2b *You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.*

This clause effectively contaminates any piece of code that links with a GPL'd program.[3] In an ordinary, statically linked program, this clause represents no problem. Many modern programs,

[1] Of course we start at zero. This is computing.

[2] This mismatch between rhetoric and action also strikes me as being very similar to the behavior that Neal Stephenson noted in Apple.[31]

[3] The GPL has also been referred to as the GPV, or General Public Virus.

however, allow various forms of dynamic linking in the form of plug-ins or loadable modules. For example:

- A loadable kernel module, linked to a GPL'd kernel at run-time.

  This situation directly affects the Linux kernel. In theory, a manufacturer cannot provide a proprietary device driver for their hardware, since it violates the terms of the GPL. In practice, special dispensation has been issued by Linus Torvalds to allow such linking; this approach has been opposed by members of the FSF.[18] This disagreement represents a problem when Torvalds dies, gets bored or is kidnaped by the aliens from *Independence Day*, who want to find out a little more about virus-proof software. At that point, the enforcement of the licensing terms is subject to the ideological whims of the "Linux Foundation" – or whatever organization takes over the Linux kernel. From a business perspective, the possibility of having your IT infrastructure declared in breach of copyright over a doctrinal dispute represents an uncomfortable risk.

  Eric Raymond has argued that there is a natural advantage to open-source device drivers.[20] This argument is valid. However, the issue at stake here is who gets to make the decisions; the GPL takes the decision away from the author of the driver and places it in the hands of someone else.

- A plugin to some GPL'd program, such as the GIMP.
- A program which runs on a GPL'd virtual machine.

These requirements are not placed on *identifiable sections of that work [which] are not derived from the Program, and can be reasonably considered independent and separate works in themselves.* Kernel modules, plugins, etc. do not obviously fall under this categorization, since they cannot be considered to be independent.

7 *If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.*

Note that the GPL essentially denies you the right to make your own decisions regarding royalty payments. Better that nobody be allowed to use the software than an inch given to the patent owner. It's pretty clear that this clause violates any notion of freedom of speech or use; the GPL ties software to a particular political view.

Another problem, not directly addressed in the GPL itself, is the inability to alter licensing arrangements after a program or library has been GPL'd. As far as I can see, this is not directly addressed in the GPL itself. There is an additional line of reasoning that seems to regard the issuer of the GPL'd software as bound by the same modification clauses (2 and 4) as the receivers of the software. Other FSF articles clearly imply that this is the intent.[27] The X/Open consortium is explicitly criticized for allowing a license that makes alterations possible.

The line of reasoning in the previous paragraph is probably both fallacious and unenforceable. It would imply, for example, that the reverse-engineering prohibitions in other licenses also apply to the creator of the software, which would make debugging a little difficult. I can't see a court taking it seriously. But if this is the case, then the FSF is indulging in a sort of legalistic smokescreen; not exactly the sort of behavior one automatically associates with a defender of freedom.[4]

[4] If it isn't the case, then the situation is hardly any better. It requires the FSF to become either a litigious bully or a paper tiger.

3 Economics

If there is something that both the FSF and I agree upon, it is that the production of software needs to be justified in terms of benefit to society. This presents a certain amount of difficulty. Benefit to society is a slippery concept and not an easily measurable quantity — unlike tractor production. In addition, since Adam Smith, the best means of deriving that benefit are not necessarily direct. Which brings me to the subject of economics.

Before starting any discussion of economics, I need to pin down what I mean by “benefit to society”. Underlying the attitude of this essay is the belief that a computer is just a machine, and the benefit of a machine is derived from its use to do things. From this point of view, the main benefit to society that software brings is that it allows users to run programs to do things that they regard as useful or entertaining. There is a clear economic component to this attitude: benefit to society can be regarded as the production of programs that users want to use. This benefit is hedged about usual common-sense provisions, of course; it’s hard to argue that virus production is of benefit to anyone other than security experts.

An alternate view regards computers as ends in themselves. I’m someone who enjoys theoretical computer science and also enjoys tinkering with my systems for the pure love of it. So this view is something that I espouse in deed, if not in word. This point of view is of benefit to society in the same way that science, art or literature is; it expands our horizons and makes us mentally richer and more cultured human beings. This view is perfectly reasonable — I also think that it is a view underlying many of the attitudes of the FSF. However, in terms of wider benefit to society, it is likely to be eclipsed by the purely utilitarian considerations of the economic viewpoint.

3.1 Intellectual Property

> Whatsoever then he removes out of the state that nature hath provided, and left it in, he hath mixed his labor with, and joined to it something that is his own, and thereby makes it his property.[15]

The implicit point of view contained in this essay is a Lockean one. Producing a piece of software requires taking the state of nature, the common heritage of software tools and techniques, and using them to fashion something new.

To the extent that programming involves labor — and thinking is certainly labor, ask any student — a piece of software is [intellectual] property. To the extent that invention requires labor, an invention is property. This state of affairs is recognized in intellectual property law, such as copyright and patent law.

Nothing, of course, prevents the creator from choosing to place the fruits of their labor in the public domain, for whatever reason they choose. Or to place it under the GPL. But the choice to do so is theirs.

3.2 The Economic Arguments of the FSF

It is worth examining the economic arguments put forward by the FSF. These arguments are largely intended to refute any benefit that non-free software production might have on society at large. Unsurprisingly, I find these arguments rather unconvincing.

In one article, economic arguments are crudely mischaracterized as a form of holding-to-ransom.[28] This article claims that the economic argument is constructed in terms of proprietary software versus no software at all. However, the economic argument is largely concerned with notions of allocation of resources. It’s more efficient, in terms of producing software that users want to use, to have a feedback loop connected to users (more on this in section 3.3.2). Specialization and division of labor suggests that allowing professionalization will lead to a greater output of higher quality software; the FSF allows professionalization, but only in a restricted environment. Ricardo’s law suggests that allowing professionals to specialize and trade will provide a greater total output of *all* economic goods, not just software.[24]

Another article cites the ready copyability of software as providing a different economic model to book publishing.[29] This argument entirely ignores the underlying costs of producing a book and considers only the costs of producing a hard copy; it also ignores the existence of the photocopier. The major investment in any creative work, be it a program, book or piece of music, lies the in the process of creation. A book needs to be written, edited, re-written, typeset, published. A program needs to be written, debugged, packaged. All of these things involve work; this is where the notion of ownership comes from, not the duplication of the final piece.

Finally, there is the question of wasted effort. An example is given of closed software requiring wasted make-work to provide a suitably adapted version.[28] This is a reasonable criticism of closed software, although not of all proprietary software (see section 3.4). However, if make-work is regarded as a waste, then the GPL forces make-work for anybody unwilling to accept the terms of the GPL; this aim is implicit in the argument for GPLing libraries.[30] If closed software imposes a social and economic cost, then so does the GPL.

3.3 Amateurs, Professionals and Patronage

The main benefit to society and the economy of such intellectual property notions as copyright and patents is the creation or encouragement of a class of professionals. Allowing somebody talented in a certain field to make a living directly from that field has a number of advantages: the most obvious is the ability of talent to concentrate on what they are good at, rather than requiring them to undertake other tasks to support themselves; additionally, specialization is permitted, leading to a feedback loop where skills are honed and improved. Prior to copyright, those wishing to be inventors, authors or other creative artists had to either find a patron or have additional means.

3.3.1 Patronage

In the past, anybody not of independent means who wanted to create something intended for general release needed to find a *patron*: a person willing to foot the bills in exchange for some intangible return. Many creator-patron relationships were very fruitful, with the patron acting as a source of inspiration to the creator. However, unless they were beings of considerable foresight, patrons intended to use their association with artists and scientists to further their own political ends.

Patronage is, today, a little more sophisticated. The general model is endowments to a university or other institution which, in turn, makes research and thought available to the public domain. The patrons derive advertising and public relations benefits from their sponsorship. Most governments recognize the benefits to be had from a stream of new ideas and research moving into the public domain and provide some patronage through funding arrangements.

This form of institute-based patronage seems to be the ideal for the FSF; a community of public-spirited developers all working towards the common good.

The first problem with this model is the restricted financing available. Endowments from commerce essentially come out of a public relations budget; necessarily limited. Support from the public purse runs into the political problem of taxation levels and control. In theory, all software could be publicly funded by using the sums spent on commercial software. The rise in taxation, even if it were to take less than the cost of commercial software would be politically unpopular. The lack of control over what software is delivered is also likely to be a point of resistance.

The second problem with this model is that of feedback. Microsoft at least tries to find out what their customers want — even if it results in obvious idiocies, such as talking paper-clips.[26] The institute-based model has no feedback mechanism — or, even worse, a feedback mechanism based on internal squabbles or the political aims of the patrons. The result is that the software produced tends to reflect the interests of the programmers, rather than the users. This is obvious in most open-source: system programs, utilities and development tools abound; applications — with the exception of every hacker's favorite, games — are harder to find. The GNU offerings are almost exclusively system and development tools.

The open-source movement has provided the impetus for another form of patronage. Companies such as RedHat or Linuxcare need free software to succeed to be successful themselves. As a result, these companies hire the producers of free software to ensure the supply, provide good public relations and provide in-house expertise for the support operations that make up the companies income. This is in addition to the general desire on the company's part to do the right thing; open-source is still a social movement.

Eric Raymond argues that this form of patronage works, in part, because the companies dispensing the patronage are leaders in the field, and thus benefit in proportion.[20] If this is true, then it also represents the break-point for this form of patronage. As the market becomes more competitive, a significant *free-rider* problem appears: companies that do not have the overhead of patronage and can offer the same services at reduced cost.[13]

3.3.2 Amateurs

The alternative to professionalism or patronage is *amateurism*: doing something for the sake of interest in doing it.

The word "amateur" has acquired a negative connotation over the years, having overtones of "half-baked" or "poorly done". This extra baggage is unfortunate; many amateurs are of the highest levels of skill and dedication,[5] and I don't intend to suggest otherwise. A notable feature of amateurs is that they can approach problems with an eclectic viewpoint which may be absent from a purely professional approach.

However, an amateur production of anything, whether it be a software package or theater production requires either independent means or another source of income and a dedication to using one's spare time in the pursuit of the production to the exclusion of other activities.[6]

Open-source software is the beneficiary of a peculiar state in the software industry: many of those who are amateur programmers by night are professional programmers by day. Linux — and the Unix approach, in general — has made amateur programming attractive. Rather than produce a large, complex program requiring a huge team to produce and maintain, small packages can be produced. Software can easily be divided into front- and back-end parts, so that functionality can be produced and tested, without the overhead of a GUI.

The upshot is that open-source, at present, gains the benefits of both amateur enthusiasm and inventiveness and professional knowledge and discipline (and income). This blessed state of affairs exists while there is a pool of professional programmers able and willing to use their spare time to produce open-source software. I would suggest that the aims of the FSF will reduce this pool enormously, and the effects will be catastrophic. Eric Raymond has argued that open-source culture is essentially a gift culture; resources are in abundance and you gain status by the bestowing of gifts on the community.[22] The absence of a large supply of well-paying professional jobs in software — more or less predicated on a large scale commercial industry — will re-introduce the economics of scarcity to the software culture.

3.4 Patents Are Your Friends

Open-source software has an enviable reputation for reliability. The usual reasons given for this reliability are that the presence of the source code allows immediate analysis and rectification of any problem and that the ability to contribute enhancements and bug-fixes vastly expands the number of people working on and contributing to a piece of code.[21]

Generally, the notions of open-source and non-proprietary software are conflated — as opposed to closed, proprietary software. However, there is no particular reason to do so. Instead, software can be categorized using two axes: an open-closed axis and a free-proprietary one.[2] The benefits accruing to open-source software are largely connected to the open-closed axis. It is the making source code available that allows the peer-review and correction feedback loop to take off.

Making software proprietary does stem the flow of contributions, of course; nobody particularly wants to contribute to someone else's profit at their own cost. To offset this effect, open, proprietary software can easily provide a renumeration model, offering payment or royalties for contributions.

There is plenty of software where a proprietary model is simply not appropriate. As Neal Stephenson says, operating systems — and system software in general — seems to be software that needs to be both open-source and free.[31] Drawing on the comments about the lack of application software in section 3.3.1, I would suggest that application software, particularly vertical application software, is not particularly well-suited to free distribution. Vertical application software (VAS) has different characteristics to those of system software:

[5] For example, one of Victoria's foremost naturalist is a professional museum registrar and an amateur naturalist.

[6] *There is a fine line between "hobby" and "mental illness".* Dave Barry

- VAS integrates knowledge from another domain into computing. The domain of system software is computers, halving the amount of knowledge required. As an example, writing medical imaging software requires a knowledge of both medicine and software engineering.
- VAS is often an order of magnitude more complex than system software.

  Good software tends to be designed to be loosely coupled and recursively structured, making it simple and maintainable. And it can be so with system software, since the domain of the application is software itself and the domain operates under the same assumptions as good software design.

  In contrast, in a domain such as finance, all financial instruments are connected by various forms of equivalence. It *is* possible to design things to make them simple and composable, but considerable insight is required.
- It has a several orders of magnitude smaller user base. Everybody — well, almost everybody — needs an operating system.
- The user base may not be computer-literate, let alone computer-proficient.
- The quantity of interpreted data is usually larger in application software than system software. This has considerable flow-on effects in areas such as data management and quality assurance.

  As an example, a large enterprise would, ideally, like to keep a central client database, so that the amount of work needed to update data is minimized. Handling such a large volume of data, often in a distributed environment, tends to impose a complex management and software architecture problem.
- With application software, application-level bugs — wrong answers, rounding errors, non-compliance with regulatory requirements — take equal precedence with system-level bugs — crashes, pointer errors, etc.

  As an example, with European Monetary Union, the way European currencies are exchanged is legally required to follow a two-step procedure, known as *triangulation*.[6] A program can run through to completion with correct looking results, yet still have wrong (and illegal) results for a few cases.

  Application-level bugs are often the result of inconsistent assumptions, such as rounding assumptions or calculation methodologies. These bugs are difficult to control, even with a centralized team. Software which has tens or hundreds or thousands of programmers, all contributing in an uncoordinated fashion, is likely to have a perpetual quality control problem.

The characteristics of VAS tends to act as a pressure against free development. The end result consists of large programs that no programmer is particularly interested in, whatever the importance is to the end-user.[7] (Notable exceptions are areas with existing bodies of amateur enthusiasts: amateur radio, astronomy, etc.) To a large extent, the patronage model breaks down, as well; the small size of the user base and the large size of the projects tends to channel patronage towards other areas. And yet VAS needs to exist, if computers are to be anything other than hobbyist toys.

None of the above suggests that VAS software needs to be closed-source. The advantages of open-source VAS are still those of any other open-source software. The problem with opening proprietary VAS is simply that the producers of that software want to protect their investment. Copyright can protect the source code easily enough. Unfortunately, any piece of VAS represents a considerable amount of investment in analysis, design and algorithmics; no VAS vendor will willingly sacrifice that investment.

Enter, to the sound of ominous music, THE PATENT. A patent allows the inventor of an idea (algorithm or nifty piece of design, in software terms) the exclusive right to the idea for a limited term, in exchange for the publication of the idea. Once the term expires, the idea passes into the public domain.

If this description sounds familiar, it's not surprising; it's a form of open-source. The arguments for patents are similar to open-source arguments, as well: open publication ensures that new ideas do not get lost, that duplicated effort is avoided and that others can examine and learn from the idea.

[7] *Every good work of software starts by scratching a developer's personal itch.*[21]

To see the benefits of patents (and copyright) in maintaining information, a historical example might help. In the era of Shakespeare, a would-be publisher only had to get hold of a manuscript, by fair means or foul, to be able to copy and print it. (The first publication of Shakespeare's sonnets followed this pattern.) Similarly, any acting troupe that could gain access to a play could perform it. As a consequence, play manuscripts were deliberately obscured and divided up, with an actor being given just his lines, along with suitable cues. The natural result of this is that many Tudor and Jacobean plays exist only in fragmentary form; we only have the works of Shakespeare today because he was well-regarded enough for a syndicate to track down most of his work and publish it in the first folio.[23, 3]

As another historical example, the technique of porcelain manufacturing was almost lost to Europe. Saxony's attempts to maintain a monopoly on porcelain production led to obsessive secrecy and the intriguing that secrecy brings. Information was kept in the player's heads and withheld for political advantage.[12]

True, patents carry an additional piece of economic baggage over and above open-source: they encourage[8] inventors to work on, and profit from, their inventions. From an economic perspective, this is a quid pro quo for having come forward in the first place, rather than keeping the invention a secret.

Secrecy may not seem very relevant to software. In the general scheme of things, true secrecy is difficult to achieve, as something, a program or a file format has be be available and is vulnerable to reverse engineering. This represents an annoyance, but not a block to the dissemination of information. To prevent dissemination, if the discoverer of a new algorithm so wishes, he or she can, to coin a word, *bureauise* the algorithm: provide it only through the services of a company to which you submit information for processing. Bureauisation can occur whenever the process is complex, but the results simple. Combinatorial problems take this form, for example optimal path computation, or the prime factoring of large numbers.[9] Bureauisation represents an obvious social and economic burden, yet it is a natural consequence of no intellectual property protection.

Ultimately, patents provide the kind of legal protection needed to allow VAS vendors to open their source. Without some sort of protection, secrecy and obscurantism, with their costs, rule. It is possible to argue that the costs of intellectual property outweigh the costs of no such protection. But I think the verdict of history is against that argument.

3.4.1 The Abuse of Patents

Patents are, very definitely, on the nose in the software community. The reasons are not very hard to see: The patenting of common knowledge; The patenting of trivialities, such as the amazon.com 1-click patent;[14] The retroactive application of patents, after something has become common practice, such as the British Telecom hyperlink patent or the LZW algorithm patent.[25, 34] All these practices represent damage done to the community. Patents also impose a burden on companies and individuals, requiring patent searches before any moderately clever software is written.

I would argue that this sort of behavior represents an abuse of the patent process. In general, patents are required to be both new and involve an inventive step — something not obvious to someone well versed in the field of the invention.[1] Trivialities, such as the 1-click patent clearly violate this rule. Under some patent laws, a patent must be actively worked to be valid.[4] The practice of obtaining a patent, allowing the technique to come into common use and then attempting to collect royalties could be prevented by an application of such laws, even though the algorithm, in the case of LZW, is clearly non-trivial, non-obvious and patentable in itself.

4 So Where Now?

So, the GPL is an attempt to restrict freedom and the economics of software production suggest that a pure free-software model will restrict access to software. Does this leave anybody any room?

Whatever I say, the FSF will continue to produce and publish high-quality, reliable software under the terms of the GPL. People will continue to use this software to produce programs — both free and commercial — for the simple reason that they are both the best tools and the terms of the GPL do not

[8] Actually, sometimes require …

[9] In theory, the proof that $P = NP$ could end up as a jealously guarded secret.

significantly impact general use. I doubt that the FSF will come clean as to their (in my opinion, rather disingenuous) specialized use of "free". In any case, Richard Stallman's reported tantrums suggest that the issue is way beyond rational debate.[16]

What I am saying is directed at potential users of the GPL: examine the terms of the GPL carefully — do you really agree with what you are buying into? More generally, it's now time for the free software movement to grow up. Recognition of the legitimacy of other models of software production is required: it's not "us vs. them" and it's not "one or the other"; it's "us and them" and it's "both one and the other". Commerce is not somehow tainted and unclean.

If people want to make software free, then they should release it into the public domain without restraint. The MIT license provides a template for such terms.[17]

Copyright and patents, although often perverted within the software industry, have a legitimate place in the world, as does the concept of intellectual property. It's up to the software community to make them work. This means respecting legitimate patent claims; it also means vigorously opposing patents for techniques in the public domain, trivial algorithms and retroactive application.

Freedom is one of those difficult words. Real freedom, as opposed to "do what thou will shall be the whole of the law", requires respect for those who wish to take a different path; it requires contributions to the community to be for the whole community, not just those deemed ideologically acceptable. It's time for software to really be free.

Acknowledgements

Thanks go to: Alison Wain (alison@charvolant.org) for indicating where wording needed to be tightened and for pointing out various holes in my arguments; David McNett (nugget@slacker.com) for providing more detail on the Linux device driver issue; Josh Parsons (josh@coombs.anu.edu.au) for telling me that I am a Lockean — all I need to do now is read some philosophy.

References

[1] IP Australia. *Applying for a Patent*.
http://www.ipaustralia.gov.au/patents/P_apply.htm.

[2] Craig Burton. Uncollapsing open source distinctions. *Linux Journal*, 1(76), August 2000. (Interview by Doc Searls).

[3] Editors. First folio. In Encyclopædia Britannica Online [5].
http://www.eb.com.

[4] Editors. Patent. In Encyclopædia Britannica Online [5].
http://www.eb.com.

[5] *Encyclopædia Britannica Online*. Enyclopædia Britannica, Inc., 2000.
http://www.eb.com.

[6] The Association for the Monetary Union of Europe. *European Monetary Union: The Legal Framework*.
http://www.amue.org/efiles/euroframe/cliffordchance.html.

[7] Free Software Foundation. *GNU General Public License*.
http://www.gnu.org/copyleft/gpl.html.

[8] Free Software Foundation. *GNU Lesser General Public License*.
http://www.gnu.org/copyleft/lesser.html.

[9] Free Software Foundation. *What is Free Software*.
http://www.fsf.org/philosophy/free-sw.html.

[10] Free Software Foundation. *Is Microsoft the Great Satan?*, 1997.
http://www.gnu.org/philosophy/microsoft.html.

[11] *Free Software Foundation*.
http://www.fsf.org.

[12] Janet Gleeson. *The Arcanum*. Bantam, London, 1998.

[13] John Gross and Deborah Antkoviak. *Public Economics — Public Goods — The Free Rider Problem*.
http://econweb.com/Sample/PublicGoods/FreeRider1.html.

[14] Perl Hartman, Jeffrey Bezos, Shel Kaphan, and Joel Spiegel. *Method and system for placing a purchase order via a communications network*. US Patent Number 5,960,411.
http://www.uspto.gov/patft/index.html.

[15] John Locke. *Two Treatises on Government*. 1764.
http://history.hanover.edu/early/locke/j-l2-001.htm.

[16] Bertrand Meyer. The ethics of free software. *Software Development*, March 2000.
http://www.sdmagazine.com/features/2000/03/f4.shtml.

[17] *The MIT License*.
http://www.opensource.org/licenses/mit-license.html.

[18] *Distilled report about potential GPL violation by MOSIX people*.
http://www.openresources.com/news/March_5_Distilled_report.html.

[19] George Orwell. *Politics and the English Language*, 1946.
http://www.abattoir.com/~prime8/Orwell/patee.html.

[20] Eric Raymond. *The Magic Cauldron*, 1999.
http://www.tuxedo.org/~esr/writings/magic-cauldron.

[21] Eric Raymond. *The Cathedral and the Bazaar*, August 2000.
http://www.tuxedo.org/~esr/writings/cathedral-bazaar/index.html.

[22] Eric Raymond. *Homesteading the Noosphere*, August 2000.
http://www.tuxedo.org/~esr/writings/homesteading/homesteading/.

[23] Kenneth Grahame Rea. theatre, history of — the elizabethan theatre. In Encyclopædia Britannica Online [5].
http://www.eb.com.

[24] David Ricardo. *The Principles of Political Economy and Taxation*. John Murray, London, 1817.
http://socserv2.socsci.mcmaster.ca:80/~econ/ugcm/3ll3/ricardo/prin/index.html.

[25] Desmond Sargent. *Information handling system and terminal apparatus therefor*. US Patent Number 4,873,662.
http://www.uspto.gov/patft/index.html.

[26] Doc Searls. The shrinking subject. *Linux Journal*, 1(76), August 2000.

[27] Richard Stallman. *Copyleft: Pragmatic Idealism*.
http://www.fsf.org/philosophy/pragmatic.html.

[28] Richard Stallman. *Why Software Should Be Free*, 1992.
http://www.fsf.org/philosophy/shouldbefree.html.

[29] Richard Stallman. *Why Software Should Not Have Owners*, 1994.
http://www.fsf.org/philosophy/why-free.html.

[30] Richard Stallman. *Why you shouldn't use the Library GPL for your next library*, February 1999. http://www.gnu.org/philosophy/why-not-lgpl.html.

[31] Neal Stephenson. *In the Beginning was the Command Line*, 1999. http://www.cryptonomicon.com/beginning.html.

[32] *TARMS, Inc.* http://www.tarms.com.

[33] *TeX Users Group*. http://www.tug.org.

[34] Terry Welch. *High speed data compression and decompression apparatus and method*. US Patent Number 4,558,302. http://www.uspto.gov/patft/index.html.