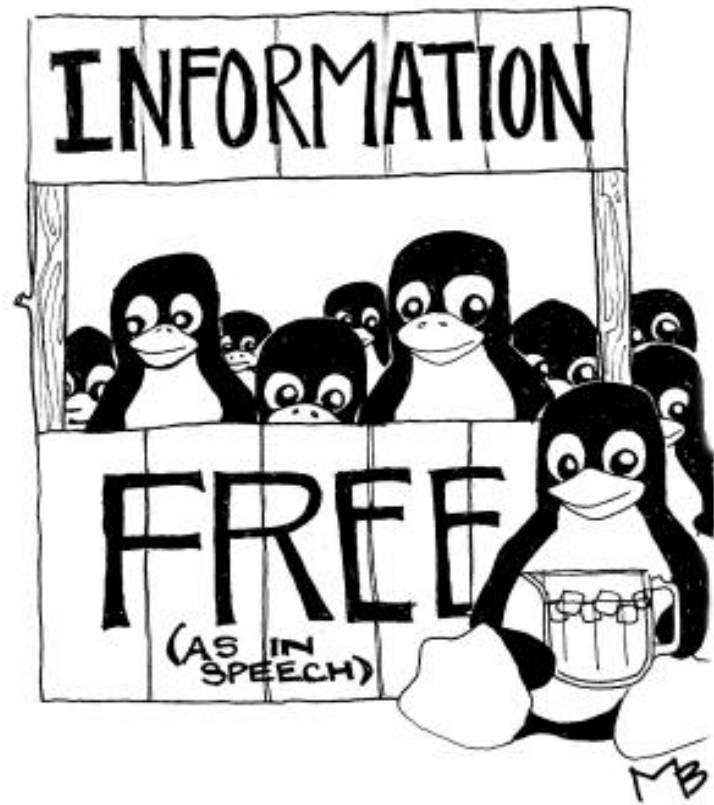# FREE SOFTWARE: HISTORY,

PERSPECTIVES, AND IMPLICATIONS

By Greg Michalec

July, 2002

FREE SOFTWARE: HISTORY,

PERSPECTIVES, AND IMPLICATIONS

Submitted to the

School of Interdisciplinary Studies

(Western College Program)

in partial fulfillment of

the requirements for the degree of

Bachelor Philosophy

Interdisciplinary Studies

by

Greg Michalec

Miami University

Oxford, Ohio

2002

<u>APPROVED</u>

Advisor_____

William H. Newell

ABSTRACT

FREE SOFTWARE: HISTORY, PERSPECTIVES, AND IMPLICATIONS

by Greg Michalec

Free software development, also known as 'open source,' is a unique phenomenon in which volunteer programmers collaboratively develop software on the Internet, and then make it available, along with its source code, for no cost. This process is particularly interesting in that many free software projects are successfully competing against traditional, proprietary products in the marketplace. This paper discusses the history and origins of free software, then considers various studies of the phenomenon from economic, sociological, and economic perspectives. Finally, I propose several possible social, political, and economic implications of free software.

ACKNOWLEDGEMENTS

Table of Contents

I. Introduction

We live in an age of increasing corporate power. The news of the latest mega-mergers is piped into our homes by giant media conglomerates. Each day, power, influence, and control become more centralized and consolidated into fewer and fewer organizations - mammoth bureaucracies that edge their way into every aspect of our lives. Technology only aids this process, and one of the most powerful entities today is the software company Microsoft. With over $40 billion in liquid assets[1], they are the untouchable of the free market – widely recognized as a monopoly, yet no one seems to have the will or the power to hold them back. Microsoft has successfully usurped IBM's place at the vanguard of the computer industry, conglomerated market share from other technology heavy-hitters, and now is managing to survive an antitrust lawsuit. Neither the free market nor the U.S. Government have been able to slow down Bill Gates' behemoth.

Yet, there is one product that is slowly eating away at Microsoft's monopoly share of the software industry. It has been sneaking in through the back doors of server rooms for years. It has no advertising budget. It has no billion-dollar war chest. The closest thing it has to a marketing campaign is an icon of a cute, smiling penguin, which may have eaten a little too much

---

[1] "This is a mind-bogglingly large pile of dough. No other nonfinancial firm has more liquid money at its disposal, and only a handful of banks do. It's more cash than Ford, ExxonMobil and Wal-Mart have combined, and nearly four times as much as Intel, the tech company with the next largest cash balance. It is enough to buy the entire airline industry -- twice. Or all the gold in Fort Knox, four times over. It is enough to buy 23 space shuttles or every major professional baseball, basketball, football and hockey team in America." Jim Frederick, "Microsoft's $40 billion bet," *CNN Money*, July 26, 2002, http://money.cnn.com/2002/04/12/pf/agenda_msft/index.htm, accessed July 30, 2002.

herring. This product - Microsoft's biggest fear - is an operating system called Linux, which is but one product of the phenomenon of Free Software.

The term 'Free Software' refers to software that has been distributed without restrictions on use. This means that it is not only available for use with no monetary cost, but that one also has access to the inner workings of the software – the source code – to make any changes, fixes, additions, or adaptations one chooses. While such software is often protected by legal licenses, these licenses generally only serve to protect the software from having further restrictions placed upon it.

A free software project constitutes an extremely complex system. The Linux kernel consists of over 3.1 million lines of code[2], and has thousands of contributors who are scattered all over the world, most of whom have never seen each other. Though English is the *lingua franca* of most large international projects, many developers only have the most rudimentary knowledge of it. Yet, collectively they are able to create a piece of software which performs millions of complex routines, runs on many different types of hardware, and allows hundreds of devices to communicate with one another – and they give it all away *for free.*

What is particularly interesting about free software, also known as 'open source software,'[3] is that it is usually developed by several individuals in collaboration with each other via the Internet, usually for no pay, yet it results in products that are of equal or greater quality to those produced in traditional software environments. This project examines what free software is, analyzes

---

[2] Generated using 'SLOCCount' by David A. Wheeler
[3] While there is some debate (discussed later) within the software community as to the specific meanings of the terms 'open source' and 'free software,' I use the terms interchangeably here.

various explanations for the phenomenon, and suggests possible implications of free software, both theoretical and practical.

Programming Basics

Before diving into this examination, it is necessary to briefly explain some of the processes and terms used in software development. The process of creating most computer programs involves two steps. The first step is to write the program in a human-readable programming language, such as C++, Java, Perl, COBOL or BASIC. A programming language provides a layer of abstraction between the logic of a program and the actual mechanics of the tasks that must be executed on a computer's hardware, though the level of abstraction depends on the language used – some languages, such as Perl or BASIC, are very close to English and are fairly easy to comprehend, while others are much more closely related to machine language. This step is known as 'programming', and its end product is known as 'source code', the human-readable documents that define a program's logic. For example, the Perl code

```
#!/bin/perl
$celsius = $ARGV[0];
$fahrenheit = ($celsius * (9/5)) + 32;
print "$celsius degrees celsius = $fahrenheit degrees
        fahrenheit.\n";
```

converts a temperature given in Fahrenheit into its Celsius equivalent, and displays the result on the screen.

The second step is known as 'compiling,' in which source code is transformed into a form that can be run on a computer. This is known as a 'binary' or 'executable' program, and can only be read by a computer, specifically only the same kind of computer that it was compiled on. A useful metaphor for programming is to think of baking a cake. The source code can be thought of as the recipe, which you can alter, add upon, or improve, before baking (compiling) it. And just as most people don't buy a recipe from the bakery, when you purchase software, you usually only purchase the executable form.[4]

Free Software Development Process

When a software developer wishes to contribute to a free software project, this is usually the process that is followed: The developer downloads the source code from a central repository. These repositories are maintained by the project itself, and often linked from a web page related to the project. This accounts for some degree of authenticity – the project members take pride in their work, and therefore would not distribute malicious code; it is up the developer to assess the trustworthiness of the project, but this is the case with all Internet downloads.

Most free software projects have at least two versions of source code available; one is deemed by the project to be 'stable' – that is, reasonably well-tested and bug free, and thus meant for public consumption. The other,

---

[4] Recipe analogy borrowed from Richard Stallman – Sam Williams. *Free as in Freedom: Richard Stallman's Crusade for Free Software*. San Francisco: O'Reilly. 2002.
     http://www.oreilly.com/openbook/freedom/ch02.html

known as 'unstable,' or 'development,' is the current state of the code as it is; this version is intended for developers or users 'at their own risk.'

Once the source has been obtained, developers may explore it and locate the area they want to change. Source code almost always contains 'comments' - side notes in plain English (or another language) describing what each piece of code is doing, notes as to the usage of particular sections, explanations of design choices, etc. which aid a developer in navigating the code. Additionally, the documentation to a piece of software is most often included with the source code. Developers may also find information on the project's website, or they may subscribe to the project's email list and ask the other developers about the code.

Once developers have made the desired change, they usually generate a file that compares their changes to the original state of the code. This file is automatically generated by a common program, and is called a 'patch.' The developer then submits the patch to the rest of the development community, usually via an email explaining the purpose of the patch (although in some larger projects the patch is given to an individual who maintains a particular section of the code).

At this point, the patch is examined for quality, to make sure it behaves as intended, keeps in line with design goals, does not negatively affect other portions of the code, etc. This is sometimes done by the consensus of the community via email, or by the aforementioned maintainers, or another individual who acts as a gatekeeper. If the change is acceptable, it is then 'merged,' or combined into the existing code, so that future developers

will download the change.

II. A Condensed History of Free Software

Pre-Software Industry

In the early days of computing, the 1960s and 70s, computers were fairly specialized, simple machines used only by highly trained professionals. In order to make the machine do anything useful, an operator would have to do a fair amount of programming. Because it was necessary to optimize one's programs to use the early hardware in the most efficient way possible, and because operating systems were not nearly as sophisticated as they are today, operating a computer required an intimate knowledge of both the underlying hardware and software. Thus, it was common practice for computer operators to share the source code of the programs they wrote.

The practice of sharing code has its roots not only in the necessity of knowing how software interacted with hardware at the most basic levels, but also from the scientific tradition most computer operators came from. The majority of early computers were used in research laboratories or the science departments of academic institutions, both of which have a strong culture of sharing results and methods within research communities.[5] It only made sense that if one had solved a particular problem, or developed a new way of performing a task, one should share that knowledge so that more work could be done.[6]

---

[5] Chris DiBona, Chris, Ockman, Sam, and Stone, Mark. "Introduction." in Chris DiBona, Sam Ockman, and Mark Stone, eds. *Open Sources: Voices from the Open Source Revolution.* Sebastopol: O'Relly, 1999. p. 2

[6] Peter Wayner. *Free for All: How Linux and the Free Software Movement Undercut the High-Tech Titans.* New York: Harper, 2000. p. 33

There was no software industry as we know it back then – most of the software used was either produced by the hardware manufacturer and shipped with the machine, or it was custom-developed in-house. The primary value lay in the machine; software was not sophisticated enough at that point to provide much value of its own.

This slowly began to change in the late 1970s and early 1980s. Hardware became powerful enough to support more sophisticated applications, and thus machines became more generic, while software was tuned to specialized tasks. With the invention of the microcomputer, computer hardware was becoming more common, and thus its price was falling. Corporations such as IBM, Digital, and AT&T anticipated that computer software would hold much greater value in coming years; thus, they sought to protect that value. They began to implement restrictive licenses and non-disclosure agreements with the software they shipped. The era of sharing software was over, and a new industry was born.

Richard M. Stallman and the GNU Project

Richard M. Stallman (often referred to simply as RMS), a computer operator at MIT's Artificial Intelligence Lab (where some of the most brilliant hackers[7] of the day worked), felt particularly affected by the change. He felt that the restrictions being placed on software

---

[7] I use the term 'hacker' as it is used within the software community – someone who uses computers and programming to identify and solve problems. The media has adopted this term to mean "someone who breaks computer security systems," while most hackers use the term 'cracker' to describe such individuals.

meant that the first step in using a computer was to promise not to help your neighbor. A cooperating community was forbidden. The rule made by the owners of proprietary software was, 'If you share with your neighbor, you are a pirate. If you want any changes, beg us to make them.'[8]

Stallman found such restrictions unacceptable. He could not continue working in an environment where he would be forced to deprive others of useful information, nor could he stand by and watch as the effects of proprietary software divided his hacker community.

Thus, in 1982, Stallman began a project that would attempt to build, from scratch, an entire operating system, as well as all of the useful programs one would require, with the intention that they would be made available for free and without restriction. He decided to base this project off the popular Unix operating system, which was designed by AT&T's Bell Labs and had been distributed under a loose academic license (AT&T then was only allowed to sell telephone service, and was using the universities to develop software to run its phone switches).[9] The primary advantage of Unix was that it was designed in such a way that it could be made to work on many different types of computers (earlier operating systems were programmed specifically for one type of machine). Stallman dubbed his project GNU, for "GNU's Not Unix," following a hacker tradition of recursive acronyms.[10]

---

[8] Richard Stallman. "The GNU Operating System and the Free Software Movement," in *Open Sources,* pg. 54

[9] Wayner, p. 34

[10] "We hackers always look for a funny or naughty name for a program, because naming a program is half the fun of writing the program. We also had a tradition of recursive acronyms, to say that the program that you're writing is similar to some existing program ... I looked for a recursive acronym for Something Is Not UNIX. And I tried all 26 letters and discovered that none of them was a word. I decided to make it a contraction. That way I could have a three-letter acronym, for Something's Not UNIX. And I tried letters, and I came across the word 'GNU.' That was it." - from Williams, http://www.oreilly.com/openbook/freedom/ch02.html

Stallman left his job at MIT in 1984 to work on the GNU project. He supported himself partially by selling media containing the code to his popular EMACS text-editing program.[11] Although the program was freely distributable, it was more convenient for some to receive a tape from Stallman through the mail than attempt to copy the code over the fledgling networks of the early 1980's. In 1985, he started the Free Software Foundation[12] (FSF), a non-profit organization to manage and support the development of free software. The term 'free software' is often misunderstood. Stallman clarifies its meaning by explaining that it refers to software that is 'Free as in speech, not as in beer.'[13] Thus, the term describes software that one is able to use freely, without limiting restrictions. There are many software programs that are available at no cost, such as freeware or shareware, yet they do not provide the same freedoms that 'free software' provides. Stallman states that a piece of software can be called 'free software' if a user has the freedom to use it for any purpose, to modify it, and to redistribute either copies of modified versions, either without cost or for a fee.[14]

Also in 1985, Stallman authored the "GNU Manifesto,"[15] which called for other hackers to join the project while championing the benefits of sharing source code. While the GNU project still has not completed its base operating system, it rapidly developed all the necessary programs and utilities needed to use an operating system effectively. Because these programs were compatible with other types of Unix (of which there were many), and were

---

[11] Stallman, p. 58
[12] http://www.fsf.org
[13] Dibona, Ockman, Stone, p. 3
[14] Stallman, p. 56
[15] http://www.gnu.org/gnu/manifesto.html

designed to run easily on many different types of computers, they quickly became popular, and gathered support for free software. Many exceeded the quality of proprietary programs that came with the various other Unixes, while others provided new services that simply did not exist in closed source software.

In 1988, Stallman and the FSF created the GNU General Public License[16] (GPL). This license, under which all software produced by the GNU project was placed (as well as many other free software projects), protected the freedoms of free software through copyright law. Its only restriction is that if one changes or uses any portion of the source code of a GPL-protected program, and then redistributes the resulting program in any way, then that program must be released under the GPL as well (thereby requiring that its source code must be made available). Thus, anyone can download the source code to any 'GPL'd' program, make some modifications to it, and sell it as their own product, *but only if they release their source code under the GPL as well.* This ensures that any improvements or alterations made to GPL software are made publicly available to the community that produced the software in the first place. The GPL "uses copyright law, but flips it over to serve the opposite of its usual purpose: instead of a means of privatizing software, it becomes a means of keeping software free."[17]

UC Berkeley and BSD

Flash back to 1977.   The University of California, Berkeley's

---

[16]   http://www.fsf.org/licenses/gpl.html
[17]   Stallman, p. 59

Computer Science Research Group had been helping to develop AT&T's Unix for years. They began to distribute free copies of their work, with source code, packaged as the "Berkeley Software Distribution," or BSD. After subsequent releases, BSD became one of the most popular operating systems available, and continued on a separate development path from AT&T's Unix offering. In 1984, as AT&T was broken up by the antitrust trials, they began to view their Unix software as a profit-generator. Thus, they required all parties with access to its source code to sign non-disclosure agreements.[18] Additionally, since BSD was based on the AT&T source code, everyone who wished to run it also had to purchase a source code license from AT&T.[19]

Members of the Berkeley group began to feel used. AT&T's intellectual property rights were restricting their work.[20] They began to separate out the code they had developed on their own, and released it separately in 1989. The license this code was released under was extremely loose: it only required that the license remain intact, and that any released product based on the BSD code must make some mention of that fact in its documentation.[21] There was no requirement to release source code, as exists in the GPL.

The initial freely-distributable release did not contain enough code to run an operating system, and thus was known as 'Networking Release 1." However, they did include important bits such as BSD's implementation of TCP/IP (Transmission Control Protocol/Internet Protocol), the networking

---

[18] Wayner, p. 34
[19] Marshall McKusick. "Twenty Years of Berkeley Unix: From AT&T-Owned to Freely Redistributable." *Open Sources,* p. 40
[20] Wayner, p. 42
[21] McKusick, p. 41

protocol used by the Internet. BSD was the first operating system to implement TCP/IP, and many other systems' use of TCP/IP has been heavily based off the open source release of the BSD code (including Microsoft's). [22]

The Berkeley group continued to identify the code in their BSD OS release that they had developed in collaboration with AT&T, and replaced it with code written by programmers on the Internet who had never seen the original code. By mid-1991, they had replaced all the necessary files except for six, and released this package as 'Networking Release 2." Within 6 months, users had written replacements for the six remaining files, and BSD was distributed as a fully free, open-source operating system.

After the initial release of Networking Release 2, a company called Berkeley Software Design Incorporated (BSDI) began developing their own replacements for the six missing files. They began selling their version of Unix in early 1992 for $995, which they advertised as a 99% discount over AT&T's system.[23] Despite the fact that there were at least two other Unix derivatives based on the BSD code base that were available for free, BSDI was sued by Unix Systems Laboratories (USL), the subsidiary of AT&T that developed and sold Unix, shortly after they began their advertising campaign.[24] USL was attempting to get an injunction against the sale of BSDI's product, claiming that it contained USL code and trade secrets.

BSDI explained that, save for the six additional files, they were simply using the code they had received from the University of California. USL then re-filed the suit against both BSDI and the University of California. USL's case

---

[22] Wayner, pp. 43-44
[23] McKusick, p. 44
[24] Wayner, p. 48

began to falter as the Berkeley group demonstrated that their code had been developed independently from AT&T's. In fact, they even managed to show that AT&T had removed the copyright notices from some of the BSD code and incorporated it into their Unix, thereby violating one of the loosest software licenses in existence.[25] The lawsuit continued for over a year, but by that point, AT&T had sold USL to Novell, who agreed to settle out-of-court. In the end, three out of 18,000 files were removed from BSD's code[26], and BSD continued to develop and distribute their software, which has split into three major free software projects: NetBSD[27], which supports as many types of computers as possible, FreeBSD[28], which concentrates on Intel-based computers, and OpenBSD[29], which focuses on security.

Linus Torvalds and Linux

In 1991, around the same time that BSD's Networking Release 2 was released, a 21-year-old university student in Finland was working on his own free operating system. Linus Torvalds began working on Linux as an experiment to learn more about operating systems and the hardware of an Intel-based machine. It started as being based on Minix, a version of Unix intended for academic purposes that one could purchase the source code to for $150.

Torvalds announced his project on a computing newsgroup in mid-

---

[25] Wayner, p. 52
[26] McKusick, p. 45
[27] http://www.netbsd.org
[28] http://www.freebsd.org
[29] http://www.openbsd.org

1991 with an email that began as follows:

> "Hello everybody out there using minix -
> I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones."[30]

Obviously not thinking much of it at first, Torvalds quickly attracted interest in his project, and more and more people began working on Linux. Participation grew quickly, partially due to the its ability to run on cheap Intel-based computers, and partially due to Linus' charisma. By early 1992, it had soon become a useful, independent operating system, and Linus released it under the GPL.

What had started as an individual hobbyist project rapidly attracted hundreds of developers and thousands of users. While initially limited to only running on Intel-based hardware, hackers from around the world wrote the code to enable Linux to run on numerous types of computers.

What is particularly interesting about Linux is that, up until that point, even free software projects were primarily developed by small, tightly-knit groups of developers who would only make code available when it was deemed 'stable.' Linux, on the other hand, was developed almost from the beginning by a large, loose collection of developers collaborating on the Internet, who shared and released code almost constantly. As self-described hacker/anthropologist Eric S. Raymond points out in his seminal work on open source software development, *The Cathedral and the Bazaar*, this method turned out to work extremely well. [31]

---

[30] Rajib Hasan. *History of Linux.* http://ragib.hypermart.net/linux/, accessed July 30, 2002.
[31] Eric Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an*

In the strict sense, Linux refers only to the Linux kernel – the very base of an operating system. In order to have a useful system, one requires many other utilities and programs. Many of these programs were provided by the GNU project, while others are the result of independent developers. (Because of Linux's extensive use of GNU software, Stallman insists that the operating system be referred to as GNU/Linux, to give credit to his project.[32]) When one downloads Linux, one usually downloads a group of these programs in addition to the Linux kernel. This is known as a 'distribution'. Not only do distributions provide a certain cluster of useful applications, but also they often have their own installation process, and method for installing and removing applications.

After only two years, groups such as Red Hat, Debian, and SuSE began sprouting up to give away – and sometimes sell – their own Linux distributions. New features were added to Linux, such as SAMBA (which was partially developed by WCP graduate John Blair), which allows Linux to transparently share files and printers with Microsoft-based networks, and clustering technology, which combines several computers to act as one (thus, one can have a very cheap, very powerful computer by combining many PC's with Linux.) 49 of the world's top 500 supercomputers are in fact clusters running the open source 'Beowulf'[33] clustering software on top of Linux.[34]

These technological advances, combined with Linux's enthusiastic user community, resulted in phenomenal growth. Many network

---

Accidental Revolutionary. San Francisco: O'Reilly, 1999. p. 30.

[32] Stallman, p. 66

[33] http://www.beowulf.org

[34] "Supercomputers getting Super-Duper," news.com, June 20, 2002. http://news.com.com/2100-1001-938032.html, accessed July 30, 2002.

administrators and information technology workers began to sneak Linux into their offices, finding the free software much more reliable and easier to administer than the proprietary solutions favored by management.

Today, Linux runs on at least 15 different types of computers, from IBM mainframes, to old Macintoshes, to Palm Pilots, and has a user base that runs well over ten million, and it is continuing to grow.[35]

Other Successful Free Software Projects

Some free software applications have become even more successful than Linux and the BSDs. One of the most outstanding free software projects is the Apache web server.[36] Begun in 1995, Apache quickly rose to become the most popular server software on the web (a web server is the software that sends web pages to a browser). According to the most recent survey, it is currently running on over 21 million machines on the Internet, capturing 56% of the market share (versus Microsoft Internet Information Server's 32%). When only active websites are considered, Apache's dominance rises to 64%, compared to Microsoft's 27%. [37]

---

[35] David Wheeler. "Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers!" http://www.dwheeler.com/oss_fs_why.html, accessed July 30, 2002.
[36] http://www.apache.org
[37] Netcraft survey – http://www.netcraft.com/survey/

Sendmail[38] is another extremely successful free software project. A 2001 survey found that Sendmail, an open source email server, is the most common email server on the Internet, at 47% of all publicly available email servers. In contrast, Microsoft's Exchange makes up only 18% of the Internet email market.[39]

PHP[40] (a recursive acronym for PHP: Hypertext Preprocessor) is a free software programming language used to create dynamic web pages. It has become the world's most popular server-side scripting program, running on over 9 million websites – 24% of all websites. PHP usage has grown 6.5% each month for the past two years.

Although it is not yet as popular as the projects mentioned above, an extremely promising free software project on the desktop is OpenOffice.org.[41] OpenOffice.org is a fully functioning open source office suite, including a word processor, spreadsheet, presentation manager, and drawing tool. Sun Microsystems released the source code to its StarOffice suite to create the OpenOffice.org project in July 2000. The project aims to reach parity with Microsoft Office, and recently had its 1.0 release in May 2002. It should be noted that this senior project was composed entirely in OpenOffice.org.

The Acceptance of Free Software by the Mainstream

As free software continued its rise within the computing community

---

[38] http://www.sendmail.org
[39] Wheeler.
[40] http://www.php.net
[41] http://www.openoffice.org

into the mid-90s, several key players in the movement began to worry that it was in need of a face-lift. Free software was seen by the conservative business community to be, at best, anti-business, and at worst, communistic. Such ideas arose mostly out of what hackers refer to as FUD (Fear, Uncertainty, and Doubt) regarding the changes that were taking place within the software industry. Nonetheless, it was felt by some that in order for the ideas of free software to truly take off within the marketplace, a public relations campaign was in order.

Thus, in the spring of 1997, a group of free software leaders, including Eric S. Raymond and publisher Tim O'Reilly got together to discuss possible solutions.[42] In addition to coming up with rebuttals and answers to counter many of the arguments and concerns often expressed by the corporate sector regarding free software, the meeting also resulted in a new term for free software: "Open Source." Along with the term, they established the criteria for a piece of software to be considered 'open source.' The 'Open Source Definition'[43] allows licenses much more loosely than the GPL, specifically in allowing proprietary use of open-source code. The Open Source Definition, like the GPL, states that software must be released with source code and without restriction. However, where the GPL requires that any derivative work or modification of GPL code must be released as GPL as well, the Open Source Definition includes licenses that allow redistribution without source, such as the BSD license. The group also set up an organization to manage the certification of various software projects, known as the Open Source

---

[42] Bruce Perens. "The Open Source Definition," *Open Sources,* p. 173
[43] http://www.opensource.org/docs/definition.php

Initiative.[44]

Stallman disapproves of the use of this term. While he considers software that meets the Open Source Definition to be free software, he disagrees with the use of the term 'open source' as opposed to 'free software' on the grounds that it de-emphasizes the aspect of freedom involved. [45]

In 1997, Eric S. Raymond wrote an analysis of free/open source software development entitled *The Cathedral and the Bazaar*. This document provided a logical explanation for how the decentralized, community-based development of open source software worked so effectively, and extolled the various benefits of this type of development, as well as free software in general. Several months after its publication, in January 1998, Netscape Corporation announced that it would be releasing its popular web browser suite, Netscape Communicator, as open source. Raymond's publication was cited as one of the primary motivators in this decision.[46]

That announcement is sometimes referred to as the "shot heard 'round the world.'"[47] For a corporation as high profile as Netscape – whose browser was the web's most popular in its early days (before it was beaten by Microsoft through what many believe were anti-competitive practices) – to release its proprietary product as open source at first seemed insane to many. Regardless, it introduced free software as a legitimate idea to the business community. The media began to publish story after story about this odd phenomenon, and soon the world knew about Linux. Though long in

---

[44] http://www.opensource.org/
[45] Stallman, p. 70
[46] Raymond, p. 203
[47] Raymond, p. 203

development, the open sourced web browser based on Netscape's code, know as Mozilla,[48] recently released the first production version of their product in June 2002.

Open source software played a very important role in the technology market explosion of the late 90s. Rising dot-coms and technology companies seeking to stretch their venture funding as far as possible made extensive use of free software solutions. For example, I personally worked for a company that employed several open source applications, running on top of Linux, in the product they marketed (which, ironically, was intellectual property management software!). Without the availability of open source applications such as Perl, MySQL, Apache, CVS, and Linux, the company would have had to close its doors much sooner than it did.

In the years that followed, free software has gained more use, legitimacy, and acceptance. In December 2000, IBM announced that it would dedicate nearly $1 billion to open source development, advertising, and services.[49] IBM, Dell, and Compaq all sell servers running Linux. Oracle now supports a version of its industry-leading, proprietary database software that runs on Linux. Apple's new OS X operating system consists of a proprietary front-end running on top of a modified, yet open source, version of BSD. Other major industry players which have either made commitments to free software include Sun Microsystems, Hewlett-Packard, AOL/Time Warner, Silicon Graphics, Motorola, Sharp Electronics, Amazon.com, and Yahoo.com.

---

[48]   http://www.mozilla.org
[49]   Wheeler

Linux can be found at gas stations[50], it has been to the North Pole[51], and has even traveled into space aboard the space shuttle.[52] It has become a leader in the processing of computer-generated images – you can now find clusters of Linux boxes crunching numbers at Dreamworks SKG, Industrial Light and Magic, and Disney, and it has been essential in creating the special effects for big-budget movies such as *Titanic*, *The Fellowship of the Ring*, and *Star Wars Episode I.*

Perhaps the best indicator of this trend is the changing response of Microsoft, one of the largest holdouts of proprietary software, and perhaps the company with the most to lose by free software's success. For a long time, Microsoft simply refused to acknowledge the existence of free software, dismissing it as hobbyist, an irrelevant trend, and wholly incompatible with business needs. Then, in October 1998, several internal Microsoft documents were leaked onto the Internet. These memos, known as the 'Halloween Documents,' address Linux and free software as a serious threat to Microsoft's business, and consider various strategies to combat it, and show that free software has the world's most powerful technology company worried.[53] Since then, Microsoft has gone from merely spreading 'Fear, Uncertainty, and Doubt' concerning free software to addressing its strengths and weaknesses as a true business threat. It has launched ad campaigns against free software, has a web page that compares Linux to its Windows 2000 server software point by point, and has recently admitted that they have

---

[50] "Q&A: Red Hat says Linux can't beat Windows," zdnet.com, February 5, 2002.
    http://zdnet.com.com/2100-1104-828802.html, accessed July 30, 2002.
[51] "Penguins Invade the North Pole," linuxdevices.com, May 4, 2002.
    http://www.linuxdevices.com/articles/AT4739871225.html, accessed July 30, 2002.
[52] "Debian Flies on the Space Shuttle," *Debian Gnu/Linux News*, April 1, 1997,
    http://www.debian.org/News/1997/19970401, accessed July 30, 2002.
[53] Perens, p.186

needed to change their business strategies to accommodate the competition generated by open source software.[54] Even more indicative of free software's acceptance and popularity is Microsoft's cynically named 'shared source' program, in which it shares small pieces of its source code with partner companies under heavily restrictive non-disclosure agreements.

While free software has been widely popular in server rooms for quite some time, only in the past few years has it gained support as a desktop operating system. Desktop window managers, such as the K Desktop Environment[55] (KDE) and GNOME[56] have made astounding advances in providing user-friendly graphical interfaces that surpass that of Microsoft's Windows in terms of features and functionality. Additionally, a wide range of open source applications allow users to write documents, burn CDs, watch movies, send instant messages, and perform many of the other tasks a common desktop user would want to do – all cost-free. It is now possible to order new computers with Linux pre-installed from Walmart.com!

Free software has come a long way since its humble beginnings in BSD and the GNU project. Sourceforge.net[57], a website that hosts free software development projects, currently lists 27,057 projects under Open Source Initiative-approved licenses, and that number increases every day.[58] It is estimated that the use of Linux alone increases 200% each year.[59] In the next section, I examine what it is about free software that makes it so

---

[54] Rich Cirillo. "Ballmer: Linux Changed our Game," *Varbusiness*.com, July 15, 2002.
       http://www.varbusiness.com/file/36355.html, accessed July 30, 2002.
[55] http://www.kde.org
[56] http://www.gnome.org
[57] http://www.sourceforge.net
[58] http://sourceforge.net/softwaremap/trove_list.php?form_cat=13
[59] http://www.opensource.org/advocacy/faq.php

compelling.

III. The Success of Free Software

The Benefits of Free Software Development

Open source software tends to out-perform its proprietary counterparts in a variety of areas, qualitative arguments for freedom aside. When examining free software, one will find that it is usually more reliable, more secure, performs better, or provides more functionality than a comparable closed source product, and often excels in more than one of these areas – not to mention that free software is almost always a better value.

These benefits certainly do not occur because open source developers are paid better! Rather, it is the process of open source development that results in higher quality software. The differences between free software development and traditional closed source development are best explained by Eric Raymond, in his previously-mentioned *The Cathedral and the Bazaar.* In it he explains how the success of Linux made him reconsider his assumptions about software development:

> Linux overturned much of what I thought I knew. I had been preaching the Unix gospel of small tools, rapid prototyping and evolutionary programming for years. But I also believed there was a certain critical complexity above which a more centralized, a priori approach was required. I believed that the most important software (operating systems and really large tools like the Emacs programming editor) needed to be built like cathedrals, carefully crafted by individual wizards or small bands of mages working in splendid isolation, with no beta to be released before its time.

Linus Torvalds's style of development – release early and often, delegate everything you can, be open to the point of promiscuity – came as a surprise. No quiet, reverent cathedral-building here – rather, the Linux community seemed to resemble a great babbling bazaar of differing agendas and approaches (aptly symbolized by the Linux archive sites, who'd take submissions from anyone) out of which a coherent and stable system could seemingly emerge only by a succession of miracles.

The fact that this bazaar style seemed to work, and work well, came as a distinct shock. As I learned my way around, I worked hard not just at individual projects, but also at trying to understand why the Linux world not only didn't fly apart in confusion but seemed to go from strength to strength at a speed barely imaginable to cathedral-builders.[60]

A famous work in software engineering states that the more people you assign to an overdue project, the more overdue it will become, as the level of complexity between the developers increases. Written by Frederick P. Brooks in 1975, it reasons that the level of complexity, difficulty in communication, and the potential for bugs increase geometrically as programmers are added to a project.[61] This idea is known as 'Brook's Law,' and, of course, the existence of Linux tends to dispute it.

Linux, and many free software projects since it, are developed by varying numbers of people, in a relatively decentralized fashion, with the primary (and often only) means of communication being email. While it's not exactly the case that it is a non-hierarchical development model, the hierarchies are much less strict than one might find in a traditional software shop, and are very much merit-based. In most larger projects, there is at

---

[60]   Raymond, pp. 29-30.
[61]   Steven Weber. "The Political Economy of Open Source Software." BRIE Working Paper. June, 2000.

least one person, possibly several people, who serve to coordinate and manage the development (such as Linux Torvalds, in the case of Linux), but due to the voluntary and open nature of open source development, they hold those positions entirely at the will of the other developers.

Raymond lists several reasons why this model is successful, and, based on his analysis, and my own experience with free software projects, I put forth the following two primary benefits of the open source development process:

Development is consumer-driven, not profit-driven

This point cannot be overstated. Because the developers of a program are often the ones who use the program the most, the features and issues they address are directly driven by consumer concerns. This practice is augmented by the fact that non-developing users have direct access to the programmers themselves when making feature requests, identifying problems, and providing feedback.

Because users have access to the source code, it is common that they will add a feature or fix a problem themselves, then contribute the change back to the primary development group. This is often how individuals become involved in open source projects – a practice Raymond identifies as "scratching a personal itch."[62]

Another custom related to this point is what Raymond identifies as "Release early, release often."[63] Open source developers make their code

---

[62] Raymond, p. 32.
[63] Raymond, p. 38.

available in good working order through 'development releases' - testing versions – very often. This allows users to try out the software and identify any new issues that may have arisen, as well as test the latest features, keeping interest in the project high. For example, the Mozilla project (the open source web browser) allows interested users to download a new test version of their open source web browser every night.

Another advantage to frequent release cycles concerns security and bug fixes. It is well known that Microsoft will delay the announcement of security problems with their software, sometimes for months, until they have a fix ready. Often the announcement is timed to manage public relations. This is not the case in the open source community – major bugs and security problems are announced immediately, and in most cases, fixes are available within days, sometimes hours, of identification.

On the flip side of the 'release often' idea, most open source projects are never released as 'stable' or 'production-ready' until they have been thoroughly verified and tested. There are no sloppy rushes to meet production deadlines, no premature shipments to coincide with TV spots.

An additional burden of profit-driven development usually not seen in free software is the forced upgrade. Much of Microsoft's business model is based on the idea that consumers will be forced to continually purchase the latest and greatest operating system in order for their new programs to run. This proves not only costly, but can be quite confusing, as users must constantly reacquaint themselves with new systems every few years. In free software, developers go to great lengths to make software backwards

compatible (new versions able to work with older versions), and to ensure that they are programming for the lowest common denominator. And, of course, when a new version does come out, it is available free of charge.

Probably one of the biggest causes for the high quality of open source software seems to be overlooked by Raymond. The developers of free software have direct control over their work, and therefore do not risk becoming alienated from their labor, as is often the case in hierarchical corporate environments. Free software hackers perform their tasks not out of material necessity, but out of internal motivation – because they enjoy solving problems, they feel empowered by the community, they wish to see better software, etc.

I once saw Linus Torvalds speak at a Linux Users Group meeting in San Francisco, at which he was asked about the technical performance of Sun's proprietary Unix versus that of Linux in some particular area. The individual wanted to know if Linus felt that Linux performed better due to technical detail A or technical detail B. Linus responded, "Those other operating systems aren't bad because of [technical detail A or technical detail B.] Those systems are bad because the people don't care."[64]

"Given enough eyes, all bugs are shallow"[65]

Another great benefit of the open source development model is that the number of people who are available to identify and correct bugs is only limited by the popularity of the project. In the closed source world, there are

---

[64] Linus Torvalds, June, 1999 (This is a paraphrase – there are no notes available from that meeting.)
[65] Raymond, p. 41.

only as many debuggers as the company can afford to pay. It would be difficult for a traditional 'cathedral-style' development process to test its software as widely, in as many different environments, as is possible with the open source model. Although it is not uncommon for a project's mailing list to receive an identified bug and its fix in the same email, Torvalds states that usually "somebody finds the problem, and somebody *else* understands it. And I'll go on record as saying that finding it is the bigger challenge."[66] With a wide user base of non-programmers testing the latest development releases, the challenge of locating these bugs is spread among many.

An important issue related to the idea of 'multiple eyes' in free software is security. Many critics point to the availability of source code as making open source software much less secure, since malevolent crackers can see exactly how the software operates in their search for security exploits. However, many in the security industry will tell you that 'security through obscurity' is no security at all. The alternative view is that, because of open source, more people who are interested in keeping the software secure can find the security exploits and fix them, whereas with closed source, the holes remain, they are just a little harder to find.

Free Software and the Marketplace

Good software development models and ideas of freedom aside, free software must still interact with the free market, and programmers still must get paid. Most free software developers are hobbyists. While they often feel

---

[66] Raymond, p. 41

very strongly about free software, they usually work on it in their spare time, while programming in closed source environments to pay the bills. Many are not professional developers at all, but network administrators or academics. Some do not even have 'tech-related' jobs.

Not all open source development is done for free. It is becoming more and more common for companies to pay programmers to work on open source software, as businesses begin to realize the potential of open source software. This may come as something of a surprise – why would a company pay someone to develop software that they cannot own?

Many companies benefit greatly from the development of free software, in a variety of ways. In most instances, programmers are not providing the company with a product so much as they are performing a service. This service provides a company with software that adds value to an additional product they sell. Even Stallman himself made money by selling his free EMACS text editor – by providing people the service of sending them a hard copy. As Peter Wayner explains in his book *Free for All*, "he decided that it wouldn't be wrong to sell copies of software or even software services as long as you didn't withhold the source code and stomp on anyone's freedom to use the source code as they wished." [67]

Red Hat,[68] a leading distribution of Linux, makes much of their revenue providing support and other service contracts to corporations and individuals running their popular release. They employ over 650 employees,[69] a significant number of whom are free software hackers, who have

---

[67] Wayner, p. 85
[68] http://www.redhat.com
[69] http://www.redhat.com/about/presscenter/presskit/fact_sheet.html

contributed extensively to 17 free software projects, including Apache, the Linux kernel, GNOME, and their widely used method of installing and managing programs, the Red Hat Package Manager. They have also developed source for at least 20 other projects.[70]

Red Hat views its software distribution as a commodity product – the actual costs and benefits between the various distributions available are negligible.[71] In order to gain the large share of the distribution market that they enjoy, they have had to focus on brand management. People perceive Red Hat Linux to be the highest quality and most convenient distribution available. The Red Hat brand is partially successful due to its considerable contributions to the free software community – people know that Red Hat has used its success to improve the community – and thus the employment of developers has the additional benefit of improving the quality of their brand.

Many trust the Red Hat brand so much that they "prefer to purchase 'Official' Red Hat Linux in a box for $50 when they could download it for free or buy unofficial CD-ROM copies of Red Hat for as little as $2."[72] By having a well-established brand, Red Hat can focus on selling that brand to new customers. "The challenge is to focus on market size, not market share. ... The more Linux users there are, the more potential customers Red Hat has for our [distribution]."[73] And, of course, the users who subscribe to Red Hat's brand are more likely to purchase Red Hat's support services.

IBM[74] primarily produces and sells servers, as well as various services

[70] http://www.redhat.com/about/community/development.html

[71] Robert Young. "Giving It Away: How Red Hat Software Stumbled Across a New Economic Model and Helped Improve an Industry," in *Open Sources.* p. 116

[72] Young, p. 117

[73] Young, p. 117

[74] http://www-124.ibm.com/developer/opensource/linux/

associated with those servers. As open source server software has developed a large user base and good reputation, IBM has realized that, by bundling its hardware and services with free software, it can sell more servers. The software adds value to its hardware and services. Thus, by improving the software – making it more secure and robust – it can improve this value. Therefore, IBM currently employs over 100 developers who have contributed code to over 40 free software projects.[75]

Besides IBM, many hardware companies have invested in free software. Major IT corporations don't invest in projects simply because they appreciate the freedom-centered philosophies behind them. Rather, they do so because they see an opportunity for profit. One such opportunity lies in the idea that software programs – especially operating systems – are a complementary product to computer hardware; that is, when one purchases new computer hardware, it is often necessary to purchase software as well. The profit motivation comes from an economic theory that states, "All else being equal, demand for a product increases when the prices of its complements decrease."[76]

Therefore, by contributing to free software projects, IBM, Hewlett Packard, and the like can force software companies such as Microsoft to compete with free (as in beer) products, thereby driving down the price of software. With cheaper software, companies are more likely to purchase more hardware. The same is true of the various support services that these and other companies provide: if the price of web servers falls, additional web

---

[75] http://www-124.ibm.com/developer/opensource/linux/patches/
[76] Joe Spolsky. "Joe on Software: Strategy Letter V." June 12, 2002.
http://www.joelonsoftware.com/articles/StrategyLetterV.html, accessed July 30, 2002.

servers will be installed, and hence demand for web server support will increase.

It is, for the most part, agreed upon that free software development is rapidly altering the software industry, and the economics of the technology market in general; many go so far as to say it is a revolution.[77] Regardless of the terminology used, there is no question that the phenomenon of open source software is forcing many to reconsider common assumptions in regards to software and the market. However, one finds drastically different interpretations of free software, coming from different perspectives. The next section will present a survey of different analyses of free software, focusing on the various perspectives from which they are written.

---

[77] Raymond, p. 197

IV. Perspectives on Free Software Development

Contending Interpretations

Free software as a social phenomenon is a highly complex occurrence, and its existence goes against many commonly held assumptions regarding motivation, social organization, and economics. Additionally, as free software has gained momentum and popularity and become a powerful force within the software industry, it has developed real-world political and economic effects with wide-reaching implications. With these ideas in mind, it is not surprising to find that there exist many different, sometimes opposing, explanations of this phenomenon.

Even those within the hacker community have drastically different interpretations of why open source exists and what it means. I have already briefly mentioned one of these debates, perhaps the strongest point of dispute within the community: is the purpose of developing open source software to create 'free' (as in speech) software, or is it simply a useful means for creating technologically 'good' software? Certainly Richard Stallman and the Free Software Foundation are proponents of the idea of free software as an exercise of rights, protecting the products of the hacker community from proprietary restrictions. It is this more political stance that often gets Stallman and his contemporaries labeled as 'communistic' and 'anti-business' (neither label holds true), and it was for these reasons that other members of the community formed the Open Source Initiative. As previously mentioned, the

OSI was meant to give free software a face-lift of sorts, and to make it clear that large sections of the community accepted business-friendly, profit generating practices. The creators of the OSI realize that the FSF is not necessarily 'anti-business,' but they believed that the political 'free as in freedom' rhetoric was inhibiting free software's acceptance in the business community. They believe that the software can stand on its' own merits:

> While [freedom and sharing-oriented arguments] are still beloved by a vocal minority in the open-source development community, experience ... has made it clear that they are unnecessary. An entirely sufficient case for open-source development rests on its engineering and economic outcomes – better quality, higher reliability, lower costs, and increased choice.[78]

On the other hand, there are many who believe that there is much more at play than simply 'good software,' and that those who might make compromises in order to gain mainstream acceptance are selling the movement short on its potential. They argue that free software began when concerned individuals began to speak about freedom, and to stop doing so "leads to trouble. Our ability to meet the challenges and threats [to free software] depends on the will to stand firm for freedom."[79] As one concerned hacker put it,

> Stallman is right, though, we aren't talking about freedom enough. Do people really spend their weekends helping annoying new people install free software because it has a more efficient development methodology? Of course not. If it were only about efficiency, hobbyists would volunteer to replace the old ballasts in companies' fluorescent lights. It's really about the freedom – not just an individual's freedom to hack, but now, a

---

[78] Raymond, p. 141
[79] Stallman, p. 69

company's freedom to decide its own future, not be tied down by proprietary licenses and [non-disclosure agreements]. We don't have to stop talking about freedom 'just to win over the suits [business leaders].' Suits need freedom too.[80]

Of course, these quotations represent the extreme of the two positions, and even the extremes recognize that this issue should not – and does not – divide the community to the point that it loses sight of its common goal – to spread the use of free, open-sourced software.

This issue is only one of the most obvious of many differing interpretations, theories, and explanations related to open source software. By examining these various perspectives, we not only get a better understanding of the free software phenomenon, and how we have gotten to this point, but, as I will show in this section, differing assumptions concerning the nature of individuals and social institutions will lead us to vastly different conclusions. I have identified six articles that provide interesting and engaging explanations of free software development. I have grouped them by the differing perspectives from which they are written: Traditional Economic, Traditional Sociological, and Radical. Note that I in no way intend to imply that these are the only perspectives that can be found examining free software.

While examining these articles, one finds that some of the same questions emerge over and over. We can assume that their frequency suggests they are central to the explanation of free software. However, each perspective emphasizes different issues related to these questions, and each

---

[80]   Don Marti, "Day Two Download," *LinuxWorld On-Line*,
       http://www.linuxworld.com/linuxworld/linuxworldtoday/lwt-behind2.html accessed July 30, 2002.

provides different answers. I have chosen four of these questions that appear most often, and especially highlight the various interpretations the three perspectives have to offer.

The first question deals with the motivations of individual free software developers: What are the factors that cause someone to invest great amounts of time and energy in participating in open source development?

The second question relates to the management of free software development. As explained previously, some of the larger free software projects, such as Mozilla and Linux, are extremely complex, and the independent developers sometimes number in the thousands. What sort of organizational structures and practices are needed to manage these projects?

The third question addresses the larger economic issues surrounding open source. As free software becomes more popular, it inevitably has an effect on the software industry, especially as it begins to compete directly with proprietary products. How does free software interact with the marketplace, and how is the proprietary software responding to open source?

The final question focuses on the future of the free software movement. As this type of software development continues, where is free software going, and what are its long-term implications?

Traditional Economic Perspective

Josh Lerner and Jean Tirole's "The Simple Economics of Open Source" sets out to account for open source development within existing theoretical economic frameworks.

The Cathedral and The Bazaar, by Eric Steven Raymond, is an ethnographic account and economic analysis of the free software community from its most prominent self-described anthropologist and libertarian.

The final two articles are similar in that they attempt to explain the free software phenomenon primarily through the traditional rules of market economics. Because they both assume human beings as rational, self-interested, and autonomous, and because they focus on individual behavior as opposed to the larger social context, I place them in a traditional economic perspective.

Lerner and Tirole, in "The Simple Economics of Open Source," begin to explain the open source developer's motivations as follows: "A programmer participates in a project ... only if she derives a net benefit [which is] equal to the *immediate* payoff (current benefit minus current cost) plus the *delayed* payoff."[81] Immediate benefits and costs include monetary compensation (if it is a commercial project), "fixing a bug or customizing a program for her own benefit,"[82] and the opportunity cost of time spent, meaning that the developer is unable to engage in other activity while working on this project.

Delayed rewards consist of the 'career concern incentive,' which

---

[81] Josh Lerner and Jean Tirole. "The Simple Economics of Open Source." NBER Working Paper. March, 2000. p. 14
[82] Lerner and Tirole, p. 14

"refers to future job offers, shares in commercial open source-based companies, or future access to venture capital,"[83] and the 'ego gratification incentive,' or peer recognition. These incentives are grouped together as the 'signaling incentive.' They then explain that, while commercial projects "have an edge on the current-compensation dimension," open source projects, "may well lower the cost for the programmer, for two reasons."[84] The 'alumni effect' results from the fact that open source software is often used at schools and university as a teaching tool, and thus programmers are often already familiar with it. Additionally, if working on the software brings about a private benefit for the developer, then, too, the cost of contribution may be lowered.

In open source development, the 'signaling incentive' is stronger than in traditional settings, for the following three reasons: Because of the open source code, it is easier to measure one's performance by directly observing the quality of one's work. The "open source programmer is her own boss and takes full responsibility for the success of a subproject."[85] Finally, the "labor market is more fluid in an open source environment,"[86] allowing developers to easily switch from project to project as they see fit.

Raymond begins his explanation of motives by stating that the "Linux world behaves in many respects like a free market or an ecology, a collection of selfish agents attempting to maximize utility...."[87] The utility developers are maximizing "is not classically economic, but is the intangible reward of their own ego satisfaction and reputation among other hackers. (One may call

---

[83]   Lerner and Tirole, p. 14
[84]   Lerner and Tirole, p. 16
[85]   Lerner and Tirole, p. 17
[86]   Lerner and Tirole, p. 18
[87]   Raymond, p. 64

their motivation 'altruistic,' but this ignores the fact that altruism is itself a form of ego satisfaction for the altruist)."[88]   The reputation gained in the free software community "can spill over into the real world in economically significant ways. It can get you a better job offer, or a consulting contract, or a book deal."[89]

Raymond folds his reputation-based analysis into the notion of a gift culture. He argues that "human beings have an innate drive to compete for social status,"[90] and that "most ways humans have of organizing are adaptations to scarcity and want. Each way carries with it different ways of gaining social status."[91]   In our exchange-based economy, "social status is primarily determined by having control of things ... to use or trade."[92] The gift culture, as opposed to other methods of organization, is an adaptation "not to scarcity but abundance."[93] In gift cultures, social status is determined by what you give away.

Raymond proposes an additional model to explain motivation, which he labels the 'craftsmanship model,' which addresses the "pure artistic satisfaction of designing beautiful software and making it work."[94] Yet he contends that these models are complementary – craftsmanship relies on reputation as a measurement of quality. Whether reputation is the developers' primary motivation or not, the reputation given by the community will affect their behavior.

---

[88] Raymond, p. 64
[89] Raymond, p. 97
[90] Raymond, p. 97
[91] Raymond, p. 98
[92] Raymond, p. 98
[93] Raymond, p. 99
[94] Raymond, p. 100

Lerner and Tirole's analysis suggests that the most important aspect of a successful free software project's organization is the presence of credible leadership. Heads of open source projects must provide a vision, ensure that the overall project is divided into smaller, independent tasks, attract other programmers, and "keep the project together"[95] (prevent it from forking[96] or being abandoned).

While in some projects there is an undisputed leader (such as Torvalds in the case of Linux), some are managed by committee through voting or consensus process. Lerner and Tirole deny the notion that "the unconstrained, quasi-anarchistic nature of the open source process leaves little scope for leadership."[97] For example, a strong vision for a project "helps coordinate expectations," while "acceptance by the leadership of a modification or addition provides somedelayed certification as to the quality of the [change] and its integration/compatibility with the overall project."[98]

A leader must win the trust of a project's developers, by assuring them that "the leader's objectives are sufficiently congruent with theirs and not polluted by ego-driven, commercial, or political biases."[99] This trust is key in preventing the project from forking into separate endeavors. It is also necessary for successful leadership to "clearly communicate its goals and evaluation procedures."[100]

Raymond explains that there are several stages of organization

---

[95] Lerner and Tirole, p. 21

[96] A 'fork' is when a software project splits to become two (or more) projects with separate code bases. It is often the result of differences in project visions, or engineering philosophies, or other disagreements, and is extremely discouraged is the free software community.

[97] Lerner and Tirole, p. 23

[98] Lerner and Tirole, p. 23

[99] Lerner and Tirole, p. 23

[100] Lerner and Tirole, p. 24

through which a free software project may advance. The simplest organizational model is the single owner/maintainer, wherein one person "makes all decisions and collects all credit and blame."[101] A larger project may employ "multiple co-maintainers working under a single 'benevolent dictator' who owns the project."[102] However, the 'benevolent dictator' does not "own his entire project unqualifiedly. Though he has the right to make binding decisions, he in effect trades away shares of the total reputation return in exchange for others' work."[103] Later, as even more contributors join a project, it may develop "two tiers of contributors: ordinary contributors and co-developers."[104] This often takes the form of individuals claiming responsibility of sub-projects within the main project, or a specific task, such as the management of bugs. In this system, the project leader "is expected to consult with those co-developers on key decisions,"[105] especially when those decisions impact an area maintained by a co-developer. Some larger projects "turn the co-developers into a 'voting committee' " or a "rotating dictatorship, in which control is occasionally passed from one member to another within a circle of senior co-developers."[106]

Lerner and Tirole discuss two primary reactions that commercial software corporations may employ in reaction to the successes of free software: they may try to "emulate some incentive features of open source processes in a distinctively closed source environment," or they could attempt "to mix open and closed source processes to get the best of both worlds."[107]

[101] Raymond, p. 123
[102] Raymond, p. 124
[103] Raymond, p. 124
[104] Raymond, p. 125
[105] Raymond, p. 125
[106] Raymond, p. 126
[107] Lerner and Tirole, p. 24

Commercial software companies are not able to employ many of the benefits of open source, such as the previously mentioned 'alumni effect,' or the permitting of users to modify and customize code. However, it is possible to duplicate some of the signaling incentives found in free software. For example, they may list the developers who have worked on a particular program. Proprietary software companies may also promote "widespread code sharing within the company."[108]

Lerner and Tirole list several strategies by which a company might "capitalize on the open source movement."[109] One such method is to provide "complementary services and products that are not supplied efficiently by the open source community."[110] In this case, the company may hire developers to work on open source. "Red Hat will make more money on support if Linux is successful.... Similarly, Sun may benefit if Microsoft's position is weakened."[111]

Another strategy is to "take a more proactive role in the development of open source software."[112] A company may release source code to a product while creating a governance structure for the process, thereby retaining some influence over the direction of development. There have also been efforts "to certify corporate open source development programs."[113] Lerner and Tirole suggest that it makes good sense to hire prominent open source developers when a company wants to benefit from open source, as the developers act as intermediaries that gain the trust of other members of the community.

---

[108] Lerner and Tirole, p. 25
[109] Lerner and Tirole, p. 26
[110] Lerner and Tirole, p. 26
[111] Lerner and Tirole, p. 27
[112] Lerner and Tirole, p. 27
[113] Lerner and Tirole, p. 28

They next discuss whether commercial activities can "pollute the open source process."[114]   One way this might occur involves a "participant who builds a valuable module and then offers proprietary [programming interfaces] to which application developers start writing."[115]   Another is that "the coexistence of commercial activities may alter the programmers' incentives."[116] They use the analogy of academic research that is sometimes influenced by commercial interests as an example of this point. Developers may "be tempted to stop interacting and contributing freely if they have an idea for a module that might yield a huge commercial payoff. Too many programmers may start focusing on the commercial side, making the open source process less exciting."[117]

Raymond begins his analysis of open source software and the software industry with a discussion of what he calls 'The Manufacturing Delusion.'[118] He argues that "most programmer-hours are spent ... writing and maintaining in-house code that has no sale value at all...."[119] That is, most software developers are paid not to work on proprietary products to be sold, but rather to create or maintain specialized code that has value only to the organization that uses it; there is no sale value for a company's internal expenditure report-generation routines, or the specific programs that display certain pieces of information from a database to their website. They are too specialized to be useful to other organizations. Raymond uses this point to claim that "software is largely a service industry operating under the persistent

[114] Lerner and Tirole, p. 30
[115] Lerner and Tirole, p. 30
[116] Lerner and Tirole, p. 30
[117] Lerner and Tirole, p. 32
[118] Raymond, p. 141
[119] Raymond, p. 143

but unfounded delusion that it is a manufacturing industry."[120] He bolsters this argument by pointing out that "the manufacturing delusion encourages price structures that are pathologically out of line with the actual breakdown of development costs."[121] 75% of the average software product's costs over its lifetime are in maintenance, debugging, and extensions.   Thus, the "common price policy of charging a high fixed purchase price and relatively low or zero support fees is bound to lead to results that serve all parties poorly."[122]

Raymond suggests that we reject the manufacturing model and replace it with "a price structure founded on service contracts, subscription, and a *continuing* exchange of value between vendor and consumer."[123] Free software has begun to force the market to adopt such a model, by exposing "what a relatively weak prop the sale value of the secret bits in closed-source software was all along."[124]

Raymond then points out that "only *sale value* is threatened by the shift from closed to open source; use value is not."[125] He goes on to describe two business models that support funding of open source projects for their use value. One involves sharing the cost of developing a particular program among many parties; the other relies on preventing the risk of a program outlasting its original developers by spreading its maintenance among a community.

There are also several business models that can profit from indirect sales value related to open source software. One such model is the hardware

---

[120] Raymond, p. 145
[121] Raymond, p. 145
[122] Raymond, p. 145
[123] Raymond, p. 147
[124] Raymond, p. 147
[125] Raymond, p. 155

vendor who release the source to software related to its hardware because the software is not its profit center. Some companies sell accessories for open source software; one such example is the book publisher O'Reilly which sells software reference manuals, and "hires and supports well known open source hackers ... as a way of building its reputation..."[126] Another model involves releasing a software product as open source, which creates a market position for services related to that product. The instance of Netscape describes a situation in which a software client was given away in the hopes that it would create a better market position for its server software. Yet another profit model describes selling software as a proprietary product, but after a certain period of time that product becomes open sourced, giving the company profits for its newest innovations, while passing the maintenance costs onto the community.[127]

Lerner and Tirole identify three primary issues that should be observed as open source development continues. The first asks, "To what extent will reliance on breaking software projects into distinct modules serve to limit the effectiveness of open source?"[128] As previously described, many open source projects are broken up into smaller, modular components to make parallel development possible. The design principles of Unix augment this technique. However, "as new open source projects move beyond their Unix origins and encounter new programming challenges, the ability to break projects into distinct units will be less possible."[129] However, they suggest that new developments in programming languages and computer science "have

---

[126] Raymond, p. 168
[127] Raymond, pp. 162-169
[128] Lerner and Tirole, p. 32
[129] Lerner and Tirole, p. 32

encouraged further modularization, and may facilitate future open source projects."[130]

The second question asks, "Can the management of open source projects accommodate the increasing number of contributors?" Currently, many open source projects receive a disproportionate number of contributions from a few developers, while most participants only make a few submissions. As these projects gain popularity and attract more contributors, it is possible that they will receive a large amount of low-quality submissions. The management of these contributions "is likely to be an increasingly onerous task, one that will swamp the efforts of a volunteer staff."[131] One solution is the model "being adapted by Sendmail, where professional employees of the for-profit firm manage the open source submissions."[132]

Finally, Tirole and Lerner ask, "To what extent do open source projects have a greater or shorter effective life span than traditional projects?"[133] A common argument is that since free software source code "is publicly available, and at least some contributions will continue to be made, its software will have a longer duration."[134] This is opposed to commercial software, which is often abandoned or no longer upgraded once the developer disappears or even if a replacement upgrade is introduced. However, "another argument is that the nature of incentives being offered open source developers ... might lead to a 'too early' abandonment of projects that experience a relative loss in popularity."[135]

[130] Lerner and Tirole, p. 32
[131] Lerner and Tirole, p. 33
[132] Lerner and Tirole, p. 33
[133] Lerner and Tirole, p. 33
[134] Lerner and Tirole, p. 33
[135] Lerner and Tirole, p. 34

Raymond's future view of open source software focuses on the question "What will the world of software look like once the open-source transition is complete?"[136] He begins by dispelling the worry that "the transition to open source will abolish or devalue [programmers'] jobs."[137] Again, "most developers' salaries don't depend on software sale value in the first place," and "there will always be plenty of work and a healthy demand for people who can make computers do things."[138]

Raymond then shifts his focus to how various sectors of the software industry will respond to open source. Infrastructure products (the Internet, the Web, and low-level communications software) will become almost entirely open source, and maintained by consortia and for-profit service companies. Applications (word processors, video games, etc.) will tend to remain mostly closed. "There are some circumstances under which the use value of an undisclosed algorithm or technology will be high enough ... that consumers will continue to pay for closed software."[139] Middleware (databases, development tools, etc.) "will be more mixed. Whether middleware categories tend to go closed or open seems likely to depend on the cost of failures, with higher cost creating market pressures for more openness."[140]

However, applications and middleware are not very stable categories; applications "tend to fall into middleware as standardized techniques develop and portions of the service are commoditized."[141] Middleware services "will in

---

[136] Raymond, p. 190
[137] Raymond, p. 191
[138] Raymond, p. 191
[139] Raymond, p. 192-193
[140] Raymond, p. 193
[141] Raymond, p. 193

turn tend to fall into the open-source infrastructure."[142] Thus, "in a future that includes competition from open source, we can expect that the eventual destiny of any software technology will be to either die or become part of the open infrastructure itself."[143] Raymond concludes that "the free market, in its widest libertarian sense including *all* un-coerced activity whether trade or gift, can produce perpetually increasing software wealth for everyone."[144]

Traditional Sociological Perspective

Steven Weber's "The Political Economy of Open Source Software" addresses questions involving the motivation of individuals, the coordination of free software development, and the complexity of open source projects. He then steps through an explanation of free software that combines microfoundations, economic logic, and socio-political structure in which he argues that the most compelling aspect of, and biggest challenge to, open source development is the coordination of projects.

Ko Kuwabara, in "Linux: A Bazaar at the Edge of Chaos," describes open source as a highly complex system which approaches chaos. He combines complexity theory with ideas from evolutionary biology to describe free software as an evolving system that employs self-organization to maintain order.

These articles are similar in that they place primary emphasis on the social structures surrounding open source development. The sociological

---

[142] Raymond, p. 193
[143] Raymond, p. 194
[144] Raymond, p. 194

perspective acknowledges that reality and perception exist within a social context, and that individuals are flexible beings subject to influence.

Kuwabara mentions several personal motivations influencing free software developers, suggesting that the "most obvious reason for anyone contributing to the Linux project is enjoyment. Programmers program because they enjoy programming; programming directly satisfies utility, and utility is its own reward."[145] However, "enjoyment is an inadequate explanation by itself in that it fails to account for the norm of cooperation and the norm of ownership."[146] Another commonly cited motivation is that individuals program free software as part of their primary job, either being explicitly employed to develop a free software project, such as a Red Hat employee, or, more commonly, in a situation where one's everyday responsibilities require open source programming, such as a network administrator.

Kuwabara poses the notion of 'anticipated reciprocity,' in which one "contribute[s] valuable information to the group in the expectation that one will receive useful help and information in return."[147] Additionally, the notion of reputation gain as a motive is brought up once again, explained in the context of gift economies: in a community driven by abundance, "the basis of social relations shift from dyadic exchange to gift giving in which social status is associated not with 'what you control but [with] what you give away.'"[148] Yet, he identifies this as somewhat problematic due to the extreme difficulty in assessing "the net benefit of doing a certain task in terms of reputation."[149] He

[145] Ko Kuwabara. "Linux: A Bazaar at the Edge of Chaos," *First Monday,* Volume 5, Number 3 (March 6th 2000), at http://www.firstmonday.dk/issues/issue5_3/kuwabara/index.html
[146] Kuwabara
[147] Kuwabara
[148] Kuwabara
[149] Kuwabara

proposes that the best explanation of motivation is the idea of 'group efficacy,' which he describes as the evolutionary process by which an actor contributes to a project if it appears successful or promising. An example is the frequent release cycles of free software projects: "frequent releases serve to stimulate and reward the contributors in the sight of their efforts realized in new kernel releases. The sense of efficacy is a function of their own satisfaction and peer approval."[150]

A final motivation Kuwabara identifies is that of identity – "the attachment to the [free software] community and the degree to which developers identify themselves with the culture of computer programmers."[151] He closes by stating that

> the evolutionary model of group efficacy drastically alters the pattern of motivation suggested by rational-choice theory. For rational actors developing Linux is only a means to an end. ...Group efficacy means that the success of the project signals new hackers to join the project and old hackers to continue contributing. The hackers contribute because they are interested in the operating system for what it is, but they continue to contribute because the Linux project also affirms the value of intellectual freedom and the norm of sharing, which are reinforced in each individual by peer reputation.[152]

In Weber's "The Political Economy of Open Source," he, too, identifies a form of internal motivation: "the fun, enjoyment, and artistry of solving interesting programming problems clearly motivates open source software developers."[153] He adds that "choosing your own work reduces perceived drudgery."[154] Also, we find reputation as a possible motivating factor in open

---

[150] Kuwabara
[151] Kuwabara
[152] Kuwabara
[153] Weber, p. 25
[154] Weber, p. 25

source development in Weber's work. He clarifies that reputation as motivation requires some sort of system in place to assure developers "that they will get appropriate credit for what they do."[155] Programmers can use this reputation for later monetary gains, "in the form of job offers, privileged access to venture capital, etc."[156] This practice will also encourage cooperation of talented programmers, "since the marginal payoff from hard work in a joint project is higher when your collaborators are as good or better than you are."[157]

However, Weber contends, "reputational concerns by themselves cannot explain successful collaboration."[158] If reputation were the primary motivating factor, then we would see more examples of hackers competing amongst themselves for more reputation, either by challenging the leadership of existing project heads (such as Torvalds), or by splitting off to form their own projects which they could lead themselves. Because this sort of competition happens very rarely ("there are no significant examples of this kind of behavior in the Linux history"[159]), there must be other factors at play. Weber identifies these as "strong elements of shared identity within the community of developers,"[160] the "set of roughly shared beliefs that make up an informal hacker culture."[161]

---

[155] Weber, p. 25
[156] Weber, p. 25
[157] Weber, p. 25
[158] Weber, p. 26
[159] Weber, p. 26
[160] Weber, p. 26
[161] Weber, p. 27

Kuwabara's explanation of the organizational processes that must occur in order to produce complex software projects draws largely from, of all places, evolutionary biology:

> Coordination is hence a crucial element sustaining collective efforts, giving the Linux project its integrity that unfolds the seemingly chaotic yet infinitely creative processes of evolution. In particular, the issue of coordination highlights an important and intriguing aspect of evolution generally overlooked: self-organization.[162]

Self-organization is the force that leads complex, evolutionary systems towards stability, through positive feedback loops that amplify certain conditions of the systems. Kuwabara argues that this principle manifests itself in the complex system of open source software development, "mak[ing] for a spontaneous order, self-imposed from the bottom up."[163]

Kuwabara extends his ideas of 'group efficacy' and reputation to describe a self-feeding cycle in which a developer is attracted to and contributes to a promising project, gaining a certain degree of reputation within that project's community. This added reputation persuades the developer to stay involved with the project, while improving the overall quality of the project, drawing in more developers.

> Reputation ... locks people into particular patterns of collaboration and interaction through reinforcement. ... Reputation opens new opportunities and reinforces existing patterns, which further consolidates reputation. And as one gains presence in the project, one is also fixed into new responsibilities as a maintainer.[164]

He concludes, "without global control from the top, the Linux project has

---

[162] Kuwabara

[163] Kuwabara

[164] Kuwabara

achieved a fine balance between evolution and self-organization, order and disorder."[165]

Weber's explanation of free software's organizational structure rests on the principles of maintaining coordination and managing complexity. Open source projects maintain coordination through a set of cultural norms. For instance, decision-making authority within a project is mandated by a norm suggesting that "the more an individual contributes to a project and takes responsibility for a piece of software, the more decision-making authority that individual is granted by the community."[166] Disputes are resolved objectively, on the basis of the shared "general conception of technical rationality," otherwise described as "let the code decide."[167] This norm is exemplified, Weber argues, by the establishment of the Open Source Initiative, which establishes "a clear priority for pragmatic technical excellence over ideology or zealotry (more characteristic of the Free Software Foundation)."[168] There exist two primary mechanisms to sanction individuals who violate the norms of a project community: 'Flaming,' which is "'public' condemnation (usually over email lists)" and 'shunning,' refusing to cooperate with [someone] after they have broken a norm."[169]

To manage the extreme complexity of software projects, Weber explains, open source developers employ 'modular design' of code, meaning "a large program works by calling in relatively small and relatively self-contained modules. Good design and engineering is about limiting the

---

[165] Kuwabara
[166] Weber, p. 30
[167] Weber, p. 31
[168] Weber, p. 31
[169] Weber, p. 32

interdependencies and interactions between modules."[170]   This "limits the reverberations that might spread out from a code change,"[171] and thus eases parallel development.   Thus, the "organizational demands on the social/political structure"[172] are reduced through engineering principles. He then brings up the idea of 'institutional isomorphism,' a sociological argument which states that "institutions that interface frequently and deeply with each other will tend to adopt similar organizational structures as a means of improving communication and reducing a broad range of transaction costs."[173] Weber proposes that this idea applies to open source in its relationship to large commercial interests. Thus, many free software projects have adopted structures that "reduce the complexity of relationships between the open source process, and the increasing range of organizations that use open source software."[174]

Kuwabara has little to say on free software's effects on the software industry. At most, he offers up New Class Theory, an idea rooted in Marxist thought which asserts that intellectuals (academics, engineers, managers, and other skilled professionals) form a new class which belongs to neither the owning nor the working class, "with distinct ideologies directly associated with their productive functions in society."[175] "The New Class believes that science and technology, rather than profit, should be the basis of productivity,"[176] and resists forms of centralized control over the processes of production for the increase of productivity and quality (I discuss New Class Theory at greater

---

[170] Weber, p. 33
[171] Weber, p. 34
[172] Weber, p. 34
[173] Weber, p. 34
[174] Weber, p. 35
[175] Kuwabara
[176] Kuwabara

length in Section V). Kuwabara links New Class Theory to the free software movement by identifying the hacker culture as a resistance against managerial control: "Hackers and programmers ... deeply resent the growing proprietary control that place legal restrictions in source codes and disrupt hacker communities through rigid division of labor and specialization under corporate management."[177] He then poses that

> in the software industry, the New Class ideology finds perhaps its purest expression in the GNU Public License.... Part legal documentation, part political manifesto, Copyleft formulates into concrete and binding terms the norm of cooperation in hacker communities that began to lose its ground against the growing tide of commercialism.

Weber's statements on the economics of open source software focus on the notion of free software as a public good. There is no way for anyone to restrict access to free software, and thus it falls into the economic category of 'non-excludability.' Additionally, anyone may download free software "without decreasing the supply that remains for others to use,"[178] so it is described as 'non-rival' as well. Normally, goods that are non-rival and non-excludable "tend to be underprovided in non-authoritative social settings."[179] This is due to the fact that individuals can use the good as a 'free rider,' with no need to contribute towards the development of the product. Why, then, does free software receive the large amount of development that it does?

Weber cites the individual motivations as part of the answer, "but there is also at play a larger economic logic."[180] Due to the nature of the

---

[177] Kuwabara
[178] Weber, p. 27
[179] Weber, p. 27
[180] Weber, p. 27

Internet, when a piece of software is made available, it is possible the product could be used millions of times, without inhibiting its original producer's use at all. To that producer, "multiple copies of this single product are not very valuable.... But single copies of multiple products are immensely valuable. "[181] Thus, "giving away a million copies of something, for at least one copy of at least one other thing ... is a good trade."[182] However, this still does not adequately explain the 'free rider' issue, as "no trade is necessary at all. It is still a narrowly rational act for any individual to take from the pot without contributing."[183] The solution, he explains, "lies in pushing the concept of non-rivalness one step further."[184] He describes free software as actually being 'anti-rival' – "the value of a piece of software to any particular user increases, as more people download and use the same software on their machines."[185] This is due to factors such as compatibility and debugging. As long as there is a core group of developers who do contribute, "the system positively benefits from free riders."[186] The previously mentioned incentives and motivations sustain the development of these core groups.

While Kuwabara makes no claims as to where free software is heading, Weber identifies four possible implications of the existence of open source. He cites free software development as evidence of Internet communication drastically reducing "the costs associated with geographically widespread collaboration."[187] This allows nearly "parallel processing of a

---

[181] Weber, p. 28
[182] Weber, p. 28
[183] Weber, p. 28
[184] Weber, p. 28
[185] Weber, p. 28
[186] Weber, p. 29
[187] Weber, p. 36

complex task," as well as "parallel distributed innovation."[188]

The second implication he suggests involves the idea of 'path dependence,' which refers to the difficulty in switching from an existing technology infrastructure to a better one, due to the massive investment required, as well as the incentives beneficiaries of the incumbent technology have to block such a switch. He states that free software, particularly Linux, "is currently challenging that equilibrium and may overturn it in the not so distant future."[189] However, it isn't clear that open source will be able to "overcome its own related deficiency of path dependence."[190] In other words, should Linux become a sub-optimal system in the future, will the community be able to make the switch to better technology?

The third idea relates to the setting of technology standards. It centers around the "politics and particularly the international politics of standards setting."[191] Open source has already shown that it has the ability to form standards around its products. However, because the community has no formal representation, and bears no allegiance even to particular countries, how will it interact with formal standards processes?

The final implication deals with the international distribution consequences of free software; namely, because it may be used by anyone in any country for no cost, it is "likely to be a powerful instrument of bootstrapping"[192] in nations which might not otherwise have been able to afford such technology. Weber adds notes of caution, but suggests that free

---

[188] Weber, p. 36
[189] Weber, p. 37
[190] Weber, p. 37
[191] Weber, p. 38
[192] Weber, p. 39

software could have a "significant impact on the international distribution of wealth."[193] (This notion is discussed further in Section V.)

Radical Perspective

Eben Moglen's "Anarchism Triumphant: Free Software and the Death of Copyright" concentrates on open source in relation to intellectual property and current copyright policies. In it, Moglen (who, incidentally, is the *pro bono* legal counsel for the Free Software Foundation) states that all forms of information, in their digital form, are virtually indistinguishable, yet different types of information carry vastly different legal implications. He then argues that free software is the first example of individuals, through the increased communicative power of the Internet, restructuring ideas of intellectual property.

Johan Söderberg's "Copyleft vs. Copyright: A Marxist Critique" discusses how Marxist theories, such as historical materialism, can be applied to free software and information in general. Because traditional Marxists view society as primarily being linked to its material base, they place little weight in the analysis of information. Söderberg defends the importance of information, citing that the current state of technology has substantially changed the value of information in relation to labor.

Both articles approach free software in ways that give great importance to power structures and social conflict, and call into question traditional economic assumptions concerning property and labor. They also

---

[193] Weber, p. 40

place less emphasis on the individual motivations and concerns of free software developers and more on the larger societal implications of free software development, and maintain a positive view of human nature. Thus, I label these articles as coming from a "radical" perspective.

Moglen addresses the motivation question by first dismissing two of the most common arguments. The first is the idea of a "hacker gift exchange culture," which he designates as 'ethnographic jargon."[194] Because free software is a commons, no exchange is enacted there:

> A few people *give away* code that others sell, use, change, or borrow wholesale to lift out parts for something else. Notwithstanding the very large number of people who have contributed to GNU/Linux, this is orders of magnitude less than the number of users who make no contribution whatever. (Emphasis added)[195]

The second answer Moglen describes as being partially right – the argument that "free software is made by those who seek reputational compensation for their activity. Famous Linux hackers, the theory is, are known all over the planet as programming deities. From this they derive either enhanced self-esteem or indirect material advancement."[196] However, the most famous developers have not done the bulk of the work: "Reputations, as Linus Torvalds himself has often pointed out, are made by willingly acknowledging that it was all done by someone else."[197] Additionally, free software often has very good documentation, yet writing documentation is not something that gains one status. Thus, while reputation motives exist,

---

[194] Eben Moglen. "Anarchism Triumphant: Free Software and the Death of Copyright," *First Monday*, Volume 4, number 8 (August 2, 1999) at
http://www.firstmonday.dk/issues/issue4_8/moglen/index.html
[195] Moglen
[196] Moglen
[197] Moglen

there must be other factors involved.

The rest of the explanation is described through the account of Vinod Vallopillil, a Microsoft employee who wrote one of the leaked internal analyses of Linux known as the "Halloween Documents." In it, he describes downloading the source code to a common networking protocol: "I'm a poorly skilled UNIX programmer but it was immediately obvious to me how to incrementally extend the DHCP client code (the feeling was exhilarating and addictive)."[198] The availability of free software allowed for an increase in Vallopilli's ability to create, which he found exhilarating. Moglen states that it is "an emergent property of human minds to create."[199] Thus individuals participate in free software development because it allows them to create in ways otherwise unavailable.

Johan Söderberg addresses similar causes in his "Copyleft vs. Copyright: A Marxist Critique."

> Basic motivations to engage in free programming are the rush of technological empowerment (Sterling, 1994), the joy of un-alienated creativity (Moglen, 1999), and the sense of belonging to a community (commonly recognized by hackers themselves as 'ego', but reputation only viable within a group of peers, i.e. A community).[200]

Yet, he places these motivations within a context of conflict, stating that they are "on a collision course with the commercial agenda of turning the Internet into a marketplace."[201]

Moglen has little to say on the organizational processes employed in

---

[198] Moglen
[199] Moglen
[200] Johan Söderberg, "Copyleft vs. Copyright: A Marxist Critique," *First Monday*, Volume 7, Number 3 (March 4, 2002) at http://www.firstmonday.dk/issues/issue7_3/soderberg/index.html
[201] Söderberg

free software development. Instead, he concentrates primarily on the power

of the Internet as a communications medium that drastically alters "the costs

of social coordination."[202]

> The development of the Linux kernel proved that the Internet
> made it possible to aggregate collections of programmers far
> larger than any commercial manufacturer could afford, joined
> almost non-hierarchically in a development project ultimately
> involving more than one million lines of computer code – a scale
> of collaboration among geographically dispersed unpaid
> volunteers previously unimaginable in human history.[203]

In explaining the free software development model, Söderberg cites

Raymond's *The Cathedral and the Bazaar*, but criticizes it as being simplistic.

> The egalitarian outlook [does not] withstand fact; ego is an
> important motivation and status hierarchies within the
> community the organizing principle. Power relations based on
> reputation, contacts, shrewdness and technical skill play an
> important role in directing any free software development. The
> anarchic ideal is further compromised by the dependency of free
> software developments on a core group of chieftains and/or a
> charismatic leader heading the project.[204]

However, he does recognize that by shifting the hierarchical coordination

structure out of the hands of capital, "capital loses dominance over labour."[205]

Moglen's article primarily deals with issues of copyright, and thus he

does not examine the way in which free software interacts with the software

industry as much as with the "intellectual property regime."[206] He argues that

the GPL, far from being an example of 'anti-commercial bias,' in fact makes

"commercial distributors of free software better competitors against

proprietary software businesses," by assuring customers that "they aren't

---

[202] Moglen
[203] Moglen
[204] Söderberg
[205] Söderberg
[206] Moglen

riding an evolutionary 'dead-end' by subscribing to a particular commercial version of Linux."[207]

Söderberg has much to say about free software's relationship with the market economy. Once again, he speaks in terms of conflict, addressing capital's use of technology to regulate social behavior. "In this power struggle resistance must increasingly be fought with technological skills. It is in this context that the hacker community and the Free Software Movement are critical."[208] He later states that "the current trend is a growing engagement with the computer underground by corporations."[209] He views the backing of open source by large technology companies as not for the "innovative capacity of the community" so much as a way to "slash labor costs."[210] He ultimately identifies a "contradiction to capital, on one hand it prospers from the technologically skilled, unpaid, social labour of users; on the other hand it must suppress the knowledge power of those users to protect the intellectual property regime."[211]

---

[207] Moglen
[208] Söderberg
[209] Söderberg
[210] Söderberg
[211] Söderberg

Moglen opens his article by arguing that all forms of information are related, and he follows that thought through in his vision of the future. "In the digital society, it's all connected. We can't depend in the long run on distinguishing one bitstream from another in order to figure out which rules apply. What happened to software is already happening to music."[212] He foresees software companies and media corporations, the "middlemen" of culture, as being "at handigrips with fate," and that "their reign is nearly done."[213]

Söderberg's analysis of the future of free software lies in the conflict between capital's attempt to maintain its influence over labor, and the hacker community's struggle to maintain their freedom. Due to the previously mentioned contradiction between capital's wish to exploit free software whilst maintaining its control of intellectual property,

> to have it both ways, capital can only rely on its hegemonic force. ...for every successful 'management' of social cooperation to boost profits, other parts of the community will be radicalized and pitched into the conflict. Inevitable, communities will turn into hotbeds of counter-hegemonic resistance.[214]

Summary of Perspectives

The various perspectives address key questions in ways that reflect their assumptions about human behavior and social structures. In regards to the personal motivation issue, the economic perspective places great significance on rational individuals weighing self-interested motivations, while

---

[212] Moglen
[213] Moglen
[214] Söderberg

the sociological perspective points to identity within a group as the primary motivation, and the writings identified with the radical perspective give much more weight to the innate creative function of human beings.

The economic approach tends to focus on the hierarchies and leadership structure of free software communities, in keeping with their focus on entrepreneurship and individualism. The sociological explanation lies in examining the importance of the groups involved, and the difficulties in managing and coordinating such highly complex projects. The radical perspective pays little attention to the organizational structure of free software projects; the articles are interested in this issue only in its relationship to larger issues at hand (the potential of the Internet, and the freedom from capitalist control).

The economic articles approach the relationship between free software and the software industry by explaining various business models that relate to open source. The two sociological articles take stances that reach beyond traditional economic explanations, emphasizing the cooperative potential of free software economics, while the radical perspective seems more interested in free software's effect on larger institutions of property and power.

Regarding the future of free software, one of the economic articles addresses its managerial challenges, while the other envisions the its future co-existence with the commercial software industry. The sociological point of view attempts to identify the broad institutional implications of free software. In the two radical articles, open source development is posed as a potential

way to alter existing power structures.

Having surveyed various research perspectives and their views on several key free software issues, the following section will propose and discuss several implications of free software development, based on my research, education, and personal experience. However, it is important to note what I have drawn from the various perspectives – for my perspective certainly influences how I view free software. I agree with the radical view in terms of motivation; while there are certainly factors such as ego, identity, and signaling incentives involved, I believe the biggest factor influencing free software developers is the urge to create. In explaining the organizational model, I look primarily towards the sociological perspective. Free software development is an amazingly complex social system, and requires a more nuanced examination than provided by either of the other two viewpoints. With regards to free software and the marketplace, I accept points from all three perspectives. The impact on power relations, the potential of cooperative economics, and the notion of software development as a service all play into my analysis. As for future implications, the following section should make my opinions clear.

V. Implications of Free Software Development

New Class Theory

One of the most compelling implications of free software development involves the previously mentioned 'New Class Theory.' Marxist class theory holds that a group of individuals is identified as a class because of their common relationship to the process of production. Human beings are, ultimately, tied to their ability to extract material resources from the environment and transform them into the items essential for life. It is through this social process that control over production of these material needs of society can allow one class to dominate another. In the traditional Marxist interpretation, there are two classes within a capitalist mode of production: the prime condition of the capitalist class, or *bourgeoisie*, is controlling the means of production; it is they who own the factories, machines, land, infrastructure, and other resources that make production possible. The second class is the proletariat, or working class; this includes any member of society that relies on the sale of labor for their well-being and survival.

This class analysis is tied to another broader theory known as "Historical Materialism." The relevant component of this theory states that, as the primary methods of production and distribution of material goods changes, so do the social relationships within a society. This, in turn, alters the power relationships between classes. Hence, as the agrarian feudal state was slowly undermined by the rising influence of markets, the power of hereditary lords

and landed nobles gave way to the mercantile class. We can see, therefore, that the manifestations of class are malleable, and linked to the various technological and social forces that influence production.

The traditional Marxist class analysis begins to lose relevance in the modern capitalist economy. The advances of the scientific, democratic, and industrial revolutions have created a production process much more efficient at creating surplus wealth, yet at the same time they make the system much more difficult to rationalize to its participants. The Frankfurt School of critical theory developed the idea that, in the modern era, there exist wide-reaching institutions whose purpose is to rationalize, legitimate, and facilitate the means of control through the production of culture and knowledge. This includes the various media, news, and advertising services, as well as academic and research institutions. All of these combine to influence the population into propagating a culture and value system that legitimates and promotes the power of the elite.

As this structure of knowledge and culture production advances, however, its participants begin to carve out their own niche within the power structure. Technology has advanced to a point that, while material production remains central to the human condition, it is knowledge and information that retains the most value within the economy. The owning class must increasingly rely on a group of skilled workers that demand higher wages and greater autonomy within their field of work. These professional knowledge workers are able to retain indirect ownership of the products of their labor through the monopolization of knowledge within their respective fields. It is

through this change that professionals begin to have a significantly different relationship to the process of production, with different goals, than either the owning or the working classes. Through their position, they are able to resist capitalist ownership and control over the knowledge they produce and the means by which they produce it, yet lack the connection to material production that is experienced by the working class, as well as reproduce and manage the systems of control that characterize capitalist class relations.

The professional maintains power by retaining ownership of knowledge through expertise within the field, and legitimates this control by pointing to notions of rationality, efficiency, quality, and technical excellence. However, these ideas are defined by the subjective terms of that particular field. It is in this sense that the attempt of capital to maintain control of information is failing; it is losing legitimacy due to the efforts of the professional class to retain control within the class system. Thus we can see the goals of the professional class (to gain autonomy, to create specialized knowledge, to retain professional ownership of that knowledge within the field, and to do these things within the structure of 'pure' rationalism) is in conflict with the goals of the capitalist, who views knowledge production as a means for continued profit-generation and hegemonic control.

As an example, consider the legal profession. Knowledge workers in this profession have managed to create great structural hierarchies within the field, including many sub-professions; they also create specialized terminology – a code language of sorts – to define issues and practices within the field. In these ways, legal professionals create and maintain a monopoly

hold on legal knowledge; they have essential carved out a portion of the knowledge economy that remains uniquely theirs. The capitalist must then draw upon members of the legal profession, and essentially submit to the structures they have created, in order to successfully navigate the legal field.

The capitalist structure has attempted to adapt to the transition from a material to knowledge economy by creating a complicated structure of intellectual property that treats information in the same way as any other piece of property. It is in this way that a body of investors can control, in the same way as any other capital good, the sale, use, and distribution of a piece of music, a word or phrase, a cure to a disease, a method of teaching, or even a genetically-altered life form. In this way, information is commodified by the demands of capital. "The contradiction that lies at the heart of the political economy of intellectual property is between the low to non-existent marginal cost of reproduction of knowledge and its treatment as scarce property."[215]

It is here that we return to free software development. Up to now, the efforts of professionals to retain autonomy in their work and ownership of their products have been individual and passive. The development of the free software movement is a case, perhaps the first, of professional knowledge workers actively working to continue the production of information while maintaining ownership within the community, for use by those with significant expertise. Through free software, they have gained autonomy in the creation of software, while, by disallowing private ownership, retaining control of the ends of their labor. Many of them – the freedom-oriented developers – have

---

[215] Christopher May, *Global Political Economy of Intellectual Property Rights: The New Enclosure?* London: Routledge, 2000. p. 42

developed a form of class consciousness in which they recognize the contradiction of their interests versus those of the owners of intellectual property, and collectively act to resist this ownership. The other group, failing to realize the power relationships involved, continues to legitimize their control by constructing arguments based on 'efficiency.'

The reason this leap has occurred within software development, as opposed to other sectors of the knowledge economy, is due to the combined effects of several circumstances. Perhaps the most influential is the fact that these workers make up the front lines in the 'digital revolution' (a tremendous change in industrial development), and thus find it much easier to redefine production practices and relationships. The previous methods of hierarchy and control are not as set within the context of the digital age. A second, related, factor is the development of the vast social network that is the Internet, which, as we have previously examined, allows for unprecedented levels of cooperation and communication. Thirdly, the attempts by capital to maintain software as property, in spite of its non-rival nature, seems particularly illogical, and thus, this obvious contradiction makes resistance all the more expected. Finally, also related to the onset of digitalization, software developers find their skills in particularly high demand. This not only allows them more power and freedom, but their high salaries, in particular, permit them to occasionally forego the wage benefits of their production.

Progressive Uses of Free Software

Another important implication of free software lies in its potential tp increase the informational infrastructure for the public good, especially in traditionally underprivileged sectors of society.

One such application is its use within public schools, and other organizations associated with civil society. Generally, these organizations are cash-strapped as it is; the increasing price of software only hurts the ability of a school to teach, of a non-profit organization to perform its day-to-day business, of awareness groups to do the necessary advertisement and reach-out. The no-cost aspect of free software eliminates expensive license fees, and could significantly decrease the overhead costs of these organizations. The typical business office environment requires no more computing tasks than word-processing, e-mail, web browsing, and the ability to share files and print services across a network. While free software isn't on the cutting edge of media playback, or gaming, or other specialized computer applications, there are dozens of methods to perform each of these day-to-day tasks within the realm of free software. In some aspects, such as stability and security, the free software applications out-perform their proprietary components.

Many free software programs, such as Linux, are also well known for their ability to successfully run on older computer hardware. This means that that an organization can purchase second hand computers – which are often auctioned off by the palette at an extremely low cost by corporations updating their stock of desktops in order to keep up with the hardware requirements of the latest version of windows. This makes the addition of computer

technology available to organizations where other methods of resource implementation would have been impossible. Underfunded public schools may now be able to teach their students the technological skills that will enhance their prospects for future employment.[216] Another possibility that we may even see community technology centers providing Internet access and other services to a community at little cost above the price of the Internet connection, the leasing of the space, and some individuals to provide basic maintenance and tutoring.[217]

Consider the example of Philadelphia schools.[218] When a teacher installed his school's only copy of Microsoft Office on several teachers' computers, Microsoft caught word and subjected the underfunded school district – all 264 schools – to a software audit to ensure that they possessed licenses for every piece of Microsoft software on every computer. Thus, the teachers in the district, whom the district could barely afford to pay, had to spend time taking inventory of every computer they could find. As a result of the negative public relations they received from the audit, Microsoft eventually 'worked out a deal' with the district, but the threat remains to any such organization.

A second positive use of free software exists in the area of international development. Open source projects, such as Linux, have an

---

[216] One might contend that individuals in such an environment would not be learning the most widely-used applications; that listing OpenOffice experience on one's resume is not as significant as familiarity with MS Office. However, I would argue that the most important skill to learn is simple exposure to them; once one gains experience with the various metaphors of a point-and-click interface, and the basic operations of a computer (i.e. Saving to a floppy, or browsing the internet, or copying a file across the network) the specific details of any application or operating system are easily learned.

[217] For more info see http://www.redhat.com/opensourcenow/ and http://www.opensourceschools.org

[218] Cave, Damien. "Microsoft to schools: Give us your lunch money!" Salon.com, July 10, 2001, http://dir.salon.com/tech/feature/2001/07/10/microsoft_school/index.html

international flavor from the outset, due to the widespread nature of their developers. It is possible to take advantage of free software in order to improve the development situation of disadvantaged communities and nations globally. Consider Steven Weber's argument:

> For this reason open source software is likely to be a powerful instrument of bootstrapping. Contrast this with the post World War II era of development economics, where debates about technology transfer to developing countries raged around the question of 'appropriate technology'. At that time, developed country governments and international development institutions were making decisions about what was 'appropriate technology' to transfer to developing countries. Open source software shifts the decision-making prerogative into the hands of people in the developing countries. Andrew Leonard implies something similar when he says "every dollar Microsoft spends attacking software piracy in the third world [sic] is a dollar of advertising for Linux and free software."[219]

Because computer infrastructure is a vital component of industrial development today, open source technology can be used to obtain near-equity with 'first-world' nations, in at least the field of software. What is of primary importance, as Weber points out, is that the use of this software can be kept completely in the hands of these communities – their access is "limited only by their own knowledge and learning, not by property rights or prices imposed on them by a developed country owner."[220] By working to close the 'technology gap' among nations, free software may have an impact on the unequal distribution of wealth and power.[221]

Finally, free software may have a significant impact on the workings of governments in relation to software. A government, like any modern-day bureaucratic organization, relies on computer and software technology for its

---

[219] Weber, p. 39
[220] Weber, p. 39
[221] For a detailed analysis of this topic, see http://danny.oz.au/freedom/ip/aidfs.html

day-to-day operations. Currently, when some governmental department requires a new piece of software, it either purchases an existing piece of software, or contracts a firm to develop it for them. Of course, the firm retains many of the ownership rights to the software, and is often able to resell it.

Free software advocates argue two points in relation to governmental use of software. Rather than purchasing software, governments should always lean towards implementing open source solutions. This, again, is primarily due to the no-cost aspect, but also to the various technical and security advantages found in free software. The second point is that when new software is to be developed with public funds, that software should be placed in the public domain via free software licenses. Rather than subsidize a firm's intellectual property development, why not contribute to the public good, so that other governmental departments, or other organizations, can partake in the benefits of software that they partially paid for?

In Peru, where dependence on proprietary software is considered another form of Yankee imperialism, the legislature is currently considering a bill that would require the use of free software throughout the government.[222] Additionally, the governments of Germany, Taiwan, and China have made moves toward implementing the widespread use free software. Studies have been conducted by the United States, the European Union, and the United Kingdom on the viability and benefits of governmental implementation of free software.[223] Open source is already used in many US governmental departments, including NASA and the FAA.

[222] "A Law to Mandate Open Source?" MSNBC.com, June 24, 2002,
http://www.msnbc.com/news/770299.asp
[223] Wheeler

Challenges to Free Software

Free software still has many challenges to face, and a number of them are of sufficiently significant magnitude to seriously limit its growth. One obstacle to the widespread use of free software involves the organization of the development process. As was brought up several times in the previous section, a free software project is an amazingly complex, yet fairly decentralized, organization. Should open source continue to rise in popularity, the number of contributors to projects, especially the most popular projects such as Linux and Apache, will surely increase as well. How will the co-developers keep up with the massive amount of submissions they will receive? How will they manage so many developers working in parallel? Will the structure of open source projects be able to handle the load?

One point to consider with these questions is that, up to this point, the technology has seemingly kept up to the task of management; that is, new tools have been developed to handle these problems when they appear. For instance, when the Mozilla was first begun, its developers received enormous amounts of bug reports from eager users testing the latest releases. In order to manage all these reports, check for duplicates, evaluate their severity, assign them to individuals for verification and resolution, etc. a number of the project's developers created a tool called 'Bugzilla.'[224] Bugzilla is an open-source, web-based bug management tool, and has proven useful to many large software projects. Another example is that the Linux kernel has recently

---

[224] http://bugzilla.mozilla.org

adopted a program called 'Bitkeeper'[225] to help manage patches to the code.[226]

Another obstacle free software must face before it is universally accepted is usability. Free software has always been developed with technical quality and engineering efficiency in mind – this is one of the reasons why it performs so well. However, this practice can make things quite difficult for novice desktop users. Because usability has taken a back door to other technical criteria, and, until recently, projects were developed for 'geek' users anyway, using some open source programs can be frustrating for the average user.

One aspect of this is integration. Free software projects rely on being modular in nature; code is separated into smaller, independent pieces not only for the technical advantages, but also out of necessity – parallel development would be extremely difficult without it. Similarly, programs often are written to do small tasks, but easily coordinate with others to perform larger tasks. This is good practice from an engineering perspective, but for a desktop user, it can make things difficult. For example, to burn a CD in Linux, one needs to have one program that sets up the CD image, and another to copy it to the CD. However, if one wants a point-and-click interface (as opposed to running these programs in a text-only command line environment), one needs a third program. The graphical component may or may not resemble the interface of the rest of the programs on one's desktop, depending on whether the program was made for the KDE desktop, or the GNOME desktop, or another desktop system. The issue of modularity leads to

---

[225] http://www.bitkeeper.com
[226] It should be noted the Bitkeeper is a closed-source program, and its use with the Linux kernel was subject to much debate.

a larger question: will free software be able to step away from its traditional engineering-centered design principles and be willing to make compromises to more user-friendly designs?

The Linux desktop, despite issues of modularity and geek-oriented software design, has come quite far in the past few years. The advances of the two primary desktop managers, which essentially have developed into mini-distributions of desktop applications, shows that there are a good number of developers out there who are dedicated to focusing on user interfaces and smooth integration of the underlying modular applications.

Perhaps the most threatening, but the least realistic, challenge to free software comes from the software giant Microsoft and its allies in the intellectual property regime. While researching this project, I ran across the following bit in Moglen's essay:

> ...Lessing warns against the false reliance, common among hackers, that information technology is inherently anarchistic. The industry is determined to re-design hardware and software to command compliance with the intellectual property regime. "Code can, and will, displace law as the primary defense of intellectual property in cyberspace." ...One suggested strategy to prevent circumvention is to bind software in hardware devices and thereby introduce a material component to the immaterial goods.[227]

After reading this quotation I remember thinking, 'that could never happen.' Shortly thereafter, I read about Microsoft's Palladium project. What Microsoft proposes is a hardware chip inside your computer that, working together with software embedded in its Windows operating system, would prevent one from running certain applications that the system deems 'insecure.' For a program

---

[227] Moglen

to be deemed 'secure,' it would need to carry a digital certificate verifying that it had not been altered in any way since the certificate was applied. Microsoft states that the primary purpose of this system is to protect the consumer from viruses and to protect their privacy. Another, not so consumer-friendly possibility, would be to enforce Digital Rights Management, ensuring that you did not watch or listen to media that has been digitally reproduced without Hollywood's permission. There are many who believe that Palladium's primary purpose is to cripple open source software. It's possible that free software projects would not be permitted to 'sign' their applications under this scheme, and, even if they were, such a signature does not combine well with the free software process: signatures would be constantly invalidated by the rapidly changing code. Should such a system become widespread enough, it might become difficult to find computers without the Palladium chip, which would become a major hindrance to free software.

It is rather unlikely that Microsoft could pull off such a scenario. They would be hard-pressed to convince consumers that such a system was developed with them in mind while removing all user control. And, as is most often is the case, such systems can always be circumvented in some way; someone will find a way around it, by either fooling the software, or disabling the hardware.

Disclaimer of Technology Benefits

I think that is important for me to explain that, despite the various benefits and potentials I have mentioned regarding free software, it is not

without its problems. Free software is merely a tool, as is any form of technology. History has shown us that such tools can be used to enslave as easily as they are used to free. The Internet has proven to be a spectacular device to enable us to communicate more easily, but at the same time, our privacy has become threatened, and we are under increasing surveillance. Additionally, digital communication seems to have come, to some degree, at the price of physical contact.

The nature of free software development is not without its problems. As stated earlier, its organization is based on hierarchy and ego; any gender analysis of the community shows that it is extremely overpopulated by men, and this fact is undoubtedly reflected in the software's development and application. The 'digital divide' is still quite wide, enormously so when considered on a global scale. It takes a great deal of privilege in order to have access to the Internet, let alone have enough surplus time to spend working on a project for which one receives no income. I merely want to point out that I do not have a utopian futurist's view of technology; our creations will always reflect the social structures that have created them.

Open Source Beyond Software

Perhaps the most intriguing point I want to consider is the possibility of applying the open source development model to areas of production besides software.

One such possibility lies in the area of medical research. It is

questionable how much longer the pharmaceutical companies can continue to restrict cheaply produced, sometimes life saving, treatments based in their claims to intellectual property. The notion that many must suffer or die because they cannot afford to pay for the knowledge a community has produced is becoming too crass for even our cynical society. Another similar application, perhaps the most glaring candidate, lies in the area of human genome research. As researchers study and discover the functions of human genetic code, it seems absurd to allow the privatization of the building blocks of human life; such information should be a public good. To restrict the process of discovering useful applications of this knowledge seems absurd; this is just the sort of innovation that the free software model would excel at promoting. However, medical research requires a great deal more infrastructure than does software development. A free research model would have to provide some manner in which to acquire the expensive materials and facilities needed.

An area in which some of the notions behind free software seem to be taking hold is that of music production. Many independent artists are finding that by sharing and distributing their creations freely via the Internet, they are developing a loyal fan base – one willing to attend concerts and pay for full albums – much more quickly than relying on the industry to do their promotion and distribution.

Another interesting application of open source principles is that of news distribution. As the major news organizations continue to become larger, their 'objectivity' becomes more and more questionable. There is a saying

among those critical of the media conglomerates: "Don't hate the media, become the media."[228] The communications infrastructure of the Internet, combined with the falling cost of recording technology, has allowed many individuals to share first-hand coverage and analysis of news events with others around the globe. The Independent Media Center[229] is a network of websites that allows visitors to post and comment on new articles in an unmoderated setting, which provide well thought editorial coverage of major events. There are over 89 affiliated indymedia centers providing local news coverage on every continent save Antarctica.

As information continues to become more valuable in our society, and knowledge producers gain more control, there certainly will be more areas in which some of the techniques used in free software development might be applied.

Intellectual Property

A final implication of free software is what its existence says about the concept of intellectual property. The notion of treating ideas as a form of property is supposed to encourage innovation, providing individuals an incentive to create new ideas. This project has shown that free software is a reaction against the commercialization of intellectual property, and that freedom from restrictive licenses, non-disclosure agreements, and other attempts to retain ownership of ideas has resulted in increased innovation and

---

[228] This quote is attributed to punk rock star and social activist Jello Biafra.
[229] http://www.indymedia.org

participation.

Let us consider how one may 'own' an idea. If I write a new program, the intellectual property regime says I may own this program, and therefore may restrict its use as I see fit, provided it is not an obvious copy of someone else's program, or uses someone else's code. But was I alone in writing the program? What about the developers who created the language the program was written in? Where did I get the idea for the program? Perhaps it was from existing programs that I thought could be improved upon. If I write a program that uses a standard graphical interface, was it my idea to use a mouse? This line of thought can be extended *ad infinitum.* Who built the computer it was written on? Did I use the Internet for ideas? What about its developers? Who provided the electricity to run my computer machines? How did I feed myself while I was writing the program? Certainly, without the help of all these people, my program could not have been written. Shouldn't they have partial ownership of my idea?

My point is that the creation of ideas is, just like any human endeavor, a highly social process. The imposition of ownership and restrictions on that process inhibits the growth and development of future ideas. And any idea that does exist will either be eventually improved upon, or die. An idea that is not shared is an idea wasted.

In many ways, the restriction of knowledge proves immoral as well as stifling. Consider the previous example of the elementary school that could not teach children properly without the word processing program; the developing nation that requires certain technologies in order to develop infrastructure; or,

perhaps the most atrocious example, thousands upon thousands of people suffering and dying from a disease because they cannot afford the high price a pharmaceutical company places on the information it owns. There is something tragically wrong with such a system that restricts the social process in such a way.

Again, I propose that intellectual property is primarily used as an attempt by those who have always benefited by owning property to extend their power into the information age. We will see continued efforts by the intellectual property regime to restrict the use of information for their own benefit, while at the same time, there will be many who resist these attempts and will work toward the further freeing of information. As media moguls, technology capitalists, and pharmaceutical giants wage war against the defenders of information, it will be a challenging fight, but I believe that, as always, history is on the side of freedom.

VI. Conclusion

In this project, I have shown that free software development is an interesting phenomena that leads us to reconsider many commonly-held assumptions regarding the nature of software, production, and intellectual property. Although there is a range of debate originating from various perspectives on the issue, it is widely viewed that free software is significantly affecting the economics and structure of the software industry, and it holds great potential to cause change in areas beyond software. Although it faces many challenges, the existence of free software raises important questions in relation to makeup of social class, the nature of knowledge production, and the feasibility of intellectual property in the digital age.

Bibliography

Chris DiBona, Sam Ockman and Mark Stone. "Introduction," *Open Sources: Voices from the Open Source Revolution.* Ed. Chris DiBona, Sam Ockman and Mark Stone. San Francisco: O'Reilly, 1999.

Rajib Hasan. *History of Linux.* Accessed July 30, 2002. http://ragib.hypermart.net/linux/

Ko Kuwabara. "Linux: A Bazaar at the Edge of Chaos," *First Monday,* Volume 5, Number 3 (March 6th 2000), at http://www.firstmonday.dk/issues/issue5_3/kuwabara/index.html

Josh Lerner and Jean Tirole. "The Simple Economics of Open Source." NBER Working Paper. March, 2000. Also at http://opensource.mit.edu/papers/Josh%20Lerner%20and%20Jean%20Triole%20-%20The%20Simple%20Economics%20of%20Open%20Source.pdf

Christopher May, *Global Political Economy of Intellectual Property Rights: The New Enclosure?* London: Routledge, 2000. p. 42

Marshall McKusick. "Twenty Years of Berkeley Unix: From AT&T-Owned to Freely Redistributable." *Open Sources: Voices from the Open Source Revolution.* Ed. Chris DiBona, Sam Ockman and Mark Stone. San Francisco: O'Reilly, 1999.

Eben Moglen. "Anarchism Triumphant: Free Software and the Death of Copyright," *First Monday*, Volume 4, number 8 (August 2, 1999) at http://www.firstmonday.dk/issues/issue4_8/moglen/index.html

Bruce Perens. "The Open Source Definition." *Open Sources: Voices from the Open Source Revolution.* Ed. Chris DiBona, Sam Ockman and Mark Stone. San Francisco: O'Reilly, 1999.

Eric Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* San Francisco: O'Reilly, 1999.

Johan Söderberg. "Copyleft vs. Copyright: A Marxist Critique," *First*

Monday, Volume 7, Number 3 (March 4, 2002) at
http://www.firstmonday.dk/issues/issue7_3/soderberg/index.html

Richard Stallman. "The GNU Operating System and the Free Software
Movement." *Open Sources: Voices from the Open Source Revolution.*
Ed. Chris DiBona, Sam Ockman and Mark Stone. San Francisco:
O'Reilly, 1999.

Peter Wayner. *Free for All: How Linux and the Free Software Movement
Undercut the High-Tech Titans.* New York: Harper, 2000.

Steven Weber. "The Political Economy of Open Source Software." BRIE
Working Paper, June, 2000. Also at
http://brie.berkeley.edu/~briewww/pubs/wp/wp140.pdf

David Wheeler. "Why Open Source Software / Free Software (OSS/FS)?
Look at the Numbers!" July 30, 2002.
http://www.dwheeler.com/oss_fs_why.html

Sam Williams. *Free as in Freedom: Richard Stallman's Crusade for Free
Software.* San Francisco: O'Reilly. 2002. Also at
http://www.oreilly.com/openbook/freedom/

Robert Young. "Giving It Away: How Red Hat Software Stumbled Across a
New Economic Model and Helped Improve an Industry." *Open Sources:
Voices from the Open Source Revolution.* Ed. Chris DiBona, Sam
Ockman and Mark Stone. San Francisco: O'Reilly, 1999.

Appendices

The GNU Free Documentation License

Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation, Inc.

59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related

matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A) Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B) List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C) State on the Title page the name of the publisher of the Modified Version, as the publisher.

D) Preserve all the copyright notices of the Document.

E) Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F) Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G) Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H) Include an unaltered copy of this License.

I) Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J) Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K) In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L) Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M) Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N) Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as

invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be

similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

The GNU General Public License

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.

59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually

obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) 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.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute

the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless

that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

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.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to

distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR

CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

The Open Source Definition

Version 1.9

The indented, italicized sections below appear as annotations to the Open Source Definition (OSD) and are not a part of the OSD.

Introduction

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

1. Free Redistribution

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

> *Rationale: By constraining the license to require free redistribution, we eliminate the temptation to throw away many long-term gains in order to make a few short-term sales dollars. If we didn't do this, there would be lots of pressure for cooperators to defect.*

2. Source Code

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

> *Rationale: We require access to un-obfuscated source code because you can't evolve programs without modifying them. Since our purpose is to make evolution easy, we require that modification be made easy.*

3. Derived Works

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

> *Rationale: The mere ability to read source isn't enough to*

support independent peer review and rapid evolutionary selection. For rapid evolution to happen, people need to be able to experiment with and redistribute modifications.

4. Integrity of The Author's Source Code

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

Rationale: Encouraging lots of improvement is a good thing, but users have a right to know who is responsible for the software they are using. Authors and maintainers have reciprocal right to know what they're being asked to support and protect their reputations.

Accordingly, an open-source license must guarantee that source be readily available, but may require that it be distributed as pristine base sources plus patches. In this way, "unofficial" changes can be made available but readily distinguished from the base source.

5. No Discrimination Against Persons or Groups

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

Rationale: In order to get the maximum benefit from the process, the maximum diversity of persons and groups should be equally eligible to contribute to open sources. Therefore we forbid any open-source license from locking anybody out of the process.

Some countries, including the United States, have export restrictions for certain types of software. An OSD-conformant license may warn licensees of applicable restrictions and remind them that they are obliged to obey the law; however, it may not incorporate such restrictions itself.

6. No Discrimination Against Fields of Endeavor

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

Rationale: The major intention of this clause is to prohibit license traps that prevent open source from being used commercially. We want commercial users to join our community, not feel excluded from it.

7. Distribution of License

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

> *Rationale: This clause is intended to forbid closing up software by indirect means such as requiring a non-disclosure agreement.*

8. License Must Not Be Specific to a Product

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

> *Rationale: This clause forecloses yet another class of license traps.*

9. The License Must Not Restrict Other Software

The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

> *Rationale: Distributors of open-source software have the right to make their own choices about their own software.*

> *Yes, the GPL is conformant with this requirement. Software linked with GPLed libraries only inherits the GPL if it forms a single work, not any software with which they are merely distributed.*