A Funny Thing Happened on the Way to the Market:

Linux, the General Public License, and a New Model for Software Innovation

by

Matt Asay

written for

Professor Larry Lessig

Stanford Law School

April, 2002

Table of Contents

A Funny Thing Happened on the Way to the Market:

Linux, the General Public License, and a New Model for Software Innovation

Introduction

Something has changed. A year ago, I wrote a paper[1] that decried the effect the GNU General Public License ("GPL") was having on Linux, specifically, and on software, generally. But a funny thing happened on Linux's way to the market. Attitudes have changed. Talk with a Sony, IBM, or any other electronics company today, and you will almost invariably find that they are looking to deploy Linux on some or all future products. I know this from experience, as I have worked full-time for a Linux software company (Lineo) for the past two years, where I have watched Linux move from an afterthought to an almost manic goal for every significant electronics company.

What changed? What happened to remove the fear of the GPL, which was perhaps the primary de-motivator for adopting Linux? This paper will explore the change in attitude and, more importantly (for me), reverse my previous position on the GPL: that it is an innovation-killer. In place of this previous perspective, I argue that the GPL actually fosters innovation by providing an organic, robust platform upon which innovation can foster, monopoly-free.

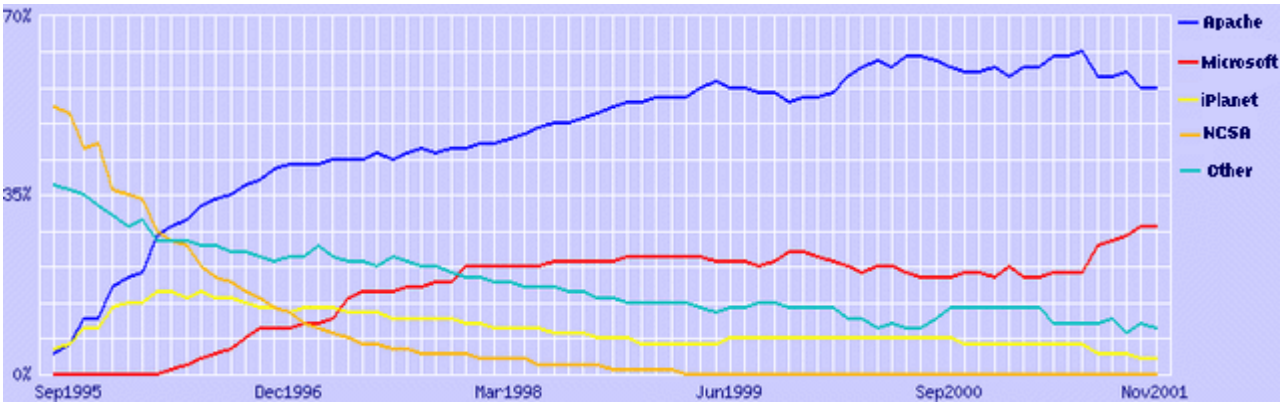
Not that Linux completely rids the world of monopoly. It just replaces Microsoft, a monopoly by one, with a global development effort, a monopoly by all, as it were (which, of course, is really no monopoly at all). This has ever been the goal: Linux and its global network of developers have long sought nothing less than "total world domination." Though uttered tongue-in-cheek by the affable

[1] Matt Asay, Linux and the Death of Innovation: The GPL and Its Economic Effects on Open-Source Software Development (May 17, 2001) (paper on file with author).

developer of the Linux kernel, Linus Torvalds, this vision is sternly enforced by the single-purpose determination of Richard Stallman, founder of the Free Software Movement ("FSF") and originator of the bulk of the Linux operating system ("OS"),[2] and the GPL that governs it.[3] Whatever one thinks of the methods of these two men, the market has opened its arms to receive them. Consider the following:

- International Data Corporation (IDC) estimates the Linux OS as having between seven to twenty-one million users worldwide, with a 200% annual growth rate.[4]

- Apache, based on open-source software, is the #1 web server on the public Internet, and has been since April 1996. Netcraft's statistics on web servers have consistently shown Apache dominating the public Internet web server market ever since Apache became the #1 web server in April 1996.



[2] Stallman argues that Linux should actually be designated "GNU-Linux" because of how much of Linux is actually GNU code that Stallman and his colleagues created. However, for the sake of clarity, the GNU-Linux operating system will be referred to as "Linux" throughout this paper. At any rate, it is not clear that volume of lines of code should determine the name of the software. *See* RUSSELL C. PAVLICEK, EMBRACING INSANITY: OPEN SOURCE SOFTWARE DEVELOPMENT 84 (2000). *See also* GLYN MOODY, REBEL CODE: INSIDE LINUX AND THE OPEN SOURCE REVOLUTION 93 (2001) (quoting Linus Torvalds, creator of the Linux kernel, who argues that "GNU-Linux" is not "catchy" enough, and that he ascribes ample credit to Stallman and the GNU code developers).

[3] For possibly the best overview of the Free Software and Open Source movements yet published, *see* MOODY, *supra* note 2.

[4] Josh Lerner and Jean Tirole, The Simple Economics of Open Source 1 (Feb. 25, 2000) (working paper on file with Stanford University's Graduate School of Business).

- GNU/Linux is the #2 server operating system sold in 1999 and 2000, and is the fastest growing. According to a June 2000 IDC survey of 1999 licenses, 24% of all servers (counting both Internet and intranet servers) installed in 1999 ran GNU/Linux. Windows NT came in first with 36%; all UNIXes combined totaled 15%…. [S]ince some of the UNIXes are [open source] systems (e.g., FreeBSD, OpenBSD, and NetBSD), the number of [such] systems is actually larger than the GNU/Linux figures….

- IDC released a similar study on January 17, 2001 titled ``Server Operating Environments: 2000 Year in Review". On the server, Windows accounted for 41% of new server operating system sales in 2000, growing by 20% - but GNU/Linux accounted for 27% and grew even faster, by 24%. Other major UNIXes had 13%.

- An Evans Data survey published in November 2001 found that 48.1% of international developers and 39.6% of North Americans plan to target most of their applications to GNU/Linux…. This is particularly surprising since only a year earlier less than a third of the international development community was writing GNU/Linux applications

- GNU/Linux had 80% as many client shipments in 1999 as Apple's MacOS. According to the June 2000 IDC survey of 1999 licenses (5.0% for Mac OS, 4.1% for GNU/Linux), there were almost as many client shipments of GNU/Linux as there were of MacOS - and no one doubts that MacOS is a client system.[5]

[5] *See* David A. Wheeler, Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers!, DWHEELER.COM WEB SITE (April 3, 2002) <http://www.dwheeler.com/oss_fs_why.html>.

The world has taken notice. Microsoft, in particular. Despite Linux's weakness in the desktop client OS market, Steve Ballmer, Microsoft's CEO, recently declared Linux to be Microsoft's chief competitive threat:

> I think one would have to rate competitors that threaten your core higher than one would rate competitors where you're trying to take from them. This puts the Linux phenomenon and the UNIX phenomenon at the top of the list. I'd put the Linux phenomenon really as threat No. 1.[6]

While some might view this as an attempt to prop up competitors in its antitrust litigation, it is more likely a realization that as non-PC-type devices proliferate (PDAs, Internet appliances, and other devices in which software is "embedded"), Microsoft's desktop dominance becomes increasingly irrelevant. Microsoft has expended considerable "ink" on the topic in its so-called "Halloween Papers," internal memos that were leaked to Eric Raymond, an Open Source advocate, and subsequently published on the Internet by Raymond. The papers analyze the Linux threat to Windows, and conclude: "OSS [Open-source software, and Linux in particular] poses a direct, short-term revenue and platform threat to Microsoft, particularly in server space.[7] Microsoft has more recently stepped up its attacks on open source, and the GPL in particular, by releasing a white paper on its new "shared source philosophy"[8] (a transparent attempt to co-opt the Open Source movement)

---

[6] Paula Rooney, *Ballmer: Linux Is Top Threat To Windows*, CMP TECHWEB (Jan. 10, 2001) <http://www.techweb.com/wire/story/TWB20010110S0006>.

[7] Vinod Valloppillil (Microsoft), *Open Source Software: A (New?) Development Methodology [Halloween I]*, OPENSOURCE.ORG, (Aug. 11, 1998) <http://www.opensource.org/halloween/halloween1.html>. *See also* Vinod Valloppillil and Josh Cohen, *Linux OS Competitive Analysis: The Next Java VM? [Halloween II]*, OPENSOURCE.ORG, (Aug. 11, 1998) <http://www.opensource.org/halloween/halloween2.html>.

[8] Microsoft's legal team, however, is quite good. Their Shared Source License is remarkable for its clarity, especially compared to the GPL. *See* Microsoft, Microsoft Shared Source License Version 1.0 for Windows CE, MICROSOFT.COM WEB SITE (April 2, 2002) <http://www.microsoft.com/windows/embedded/ce.net/previous/downloads/source/license.asp>.

and by having one of its senior vice presidents, Craig Mundie, openly deride the GPL as inimical to innovation and copyright law.[9]

At the heart of Microsoft's attacks on Linux is its derision of the General Public License ("GPL"), the source of Linux's phenomenal growth in the server market, as well as the likely cause of its frustrated progress on the client-side of the OS market. At once Linux's greatest strength and weakness, the GPL has proven to be a constant thorn in the side of Linux's market acceptance. As the argument goes, through restrictive license terms that require all derivative works to be open sourced,[10] the GPL effectively kills the ability to earn a competitive rate of return from designing GPL code, in turn diminishing the pool of developers willing to write GPL software. Though the GPL may well be enforceable in a court of law,[11] it seems doomed to unenforceability in the court of economic viability.

Unfortunately, history is increasingly unkind to this interpretation of the GPL. A year ago, Mundie and the Microsoft crew were correct in their assessment, because a year ago the world was listening. This paper will show that the world has, in fact, changed, and that the GPL, now increasingly understood (and hence, less feared), is driving Linux onto every "embedded" device and server in the world.

---

[9] Craig Mundie, *The Commercial Software Model* (May 3, 2001) (speech given at The New York University Stern School of Business, on file with Matt Asay).

[10] Or, provided with their source code, and not binary code, intact. Most software is distributed in binary form only. This allows a computer to read the hardly (humanly) decipherable sequence of 1s and 0s that make up the software code, without the software provider revealing the intellectual property driving the program. The GPL, as will be explained in more detail, requires software to be distributed with its source code, thereby enabling the user (and all third parties) to easily read the code that makes up the program. In sum, there is no such thing as intellectual property in the Open Source world, since all code must be open to the view of the user, who is free to modify the code.

[11] For arguments in favor of the GPL's enforceability, see generally Patrick K. Bobko, *Linux and General Public Licenses: Can Copyright Keep "Open Source" Software Free?*, AIPLA QUARTERLY J. (Winter 2000); Ira V. Heffen, *Copyleft: Licensing Collaborative Works in the Digital Age*, 49 STAN. L.R. 1487 (July 1997); Teresa Hill, *Fragmenting the Copyleft Movement: The Public Will Not Prevail,* UTAH L.R. 797, 812 (1999); *Shawn* W. Potter, *Opening Up to Open Source*, VI RICH. J.L. & TECH. §49 (Spring 2000); DONALD K. ROSENBERG, OPEN SOURCE: THE UNAUTHORIZED WHITE PAPERS 105-106 (2000). *But see* Dennis E. Powell, *Determining the Legality of the GPL*, LINUX PLANET, (June 26, 2000) <http://www.linuxplanet.com/linuxplanet/reports/2000/1/> (holding that the GPL's parentage, manner of transmission from user to user, its potentially contractual nature, and other issues cast doubt on the GPL's enforceability).

What Is the GPL?[12]

The GPL, or GNU General Public License, is a software license that attempts to turn copyright law on its head. Proponents dub this style of license a "copyleft" license. The GPL, which governs the Linux kernel and much open-source software, allows users to copy, modify, and distribute GPL code, with the only condition being that the user then license the derivative work under the same terms. That is, the user must GPL their derivative works.[13] Furthermore, users must agree to not establish proprietary rights in the software; to provide the source code (or access to the source code) along with the object code; [14]to include in the software notice that the code is subject to the GPL; and to accept the GPL code without warranties of any kind. Perhaps more than in any other style of software licensing, the GPL plays *the* central role in defining permissible uses of code licensed under it. It is therefore crucial to delve into the details of the GPL and fully understand them before moving on to the effects these terms have on innovation in the software industry generally, and on Linux software specifically.

---

[12] The LGPL, or Library General Public License, will not be covered in this paper. The LGPL reads much like the GPL, with the one key differentiator being that the LGPL "permits use of the library in proprietary programs [whereas] using the ordinary GPL for a library makes it available only for free programs." Free Software Foundation, *Why You Shouldn't Use the Library GPL for Your Next Library*, FSF WEB SITE (June 23, 2000) <http://www.fsf.org/philosophy/why-not-lgpl.html>. In other words, LGPL code (libraries, specifically) can link with proprietary programs without requiring that those proprietary programs be open sourced, but GPL code cannot be merged with proprietary software without that proprietary software being open sourced.

[13] *See also* PAVLICEK, *supra* note 2, at 164. The crux of the GPL centers on its definition of a derivative work, and the restrictions that attach to derivative works. The courts have (rather unsuccessfully) attempted to resolve the definition of a derivative software product, but the issue becomes even cloudier when dealing with the GPL. As will be discussed below, the GPL's definition of modified or derivative works may be more inclusive than the courts'. However, because law is as much perception as it is reality, the world's perception of the GPL (that there is a good possibility that its hyper-restrictive terms will be held to be enforceable in the courts) matters, and is what will inform the findings of this paper.

[14] Software presents a quandary: Computers do not speak human language, and most humans do not speak computer language (i.e., the binary language of 1's and 0's that informs a computer when to turn on and off circuits, etc.) So, human programmers write software in "source code," i.e., in a programming language like C++ that is readily understood by a trained programmer, which code is then "translated" (or compiled) into "object code," which is the language the computers understand. *See* WhatIs.com, Source Code Definition, WHATIS.COM WEB SITE (April 2, 2002) <http://whatis.techtarget.com/definition/0,289893,sid9_gci213030,00.html>. *See also* Bruce Abramson, Promoting Innovation in the Software Industry: A First Principles Approach to Intellectual Property Reform, 8 B.U.J. SCI. & TECH. L. 75, 114-117.

Despite Richard Stallman's unfortunate predilection for the word "free" (as in "free software"), the GPL does not require software to be provided free of charge. As the Preamble to the GPL states: "When we speak of free software, we are referring to freedom, not price."[15] The GPL further notes: "You may charge a fee for the physical act of transferring a copy, and one may at your option offer warranty protection in exchange for a fee."[16] Though the wording here is not exceptionally clear, elsewhere in the FSF's online materials Richard Stallman indicates that "transferring a copy" means the same thing as what is normally considered "selling software."[17] What is clear, then, is that one can charge for GPL software, with explicit sanction from the Free Software Foundation.

Of course, the ability to charge is somewhat diluted by the fact that GPL code is open to everyone. As noted by Lisa Green and Heather Meeker, attorneys with Wilson Sonsini Goodrich & Rosati:

> Once the copy is distributed, any recipient can distribute copies free of charge – which means your client is less likely to collect license fees, because your client is no longer the sole source. Releasing code under the GPL is not exactly like dedicating the work to the public domain…. But for business purposes, it is essentially the same. All the commercial rights (use, distribution, and modification) are licensed to anyone without charge.[18]

---

[15] Free Software Foundation, GNU General Public License, Preamble, FSF WEB SITE (June 23, 2000) <http://www.gnu.org/copyleft/gpl.html>.

[16] *Id.* *See also* Free Software Foundation, *GNU General Public License, Terms and Conditions for Copying, Distribution and Modification*, FSF WEB SITE, at §1 (June 23, 2000) <http://www.gnu.org/copyleft/gpl.html>; Richard Stallman, *The GNU Manifesto*, FSF WEB SITE (June 23, 2000) <http://www.fsf.org/gnu/manifesto.html#r1>. However, the FSF is adamant that parties using GPL derivatives cannot put a price on the source code. The FSF wants to avoid the situation were binaries (object code) are distributed without the source code, and a high access fee is charged for the source code. This would be tantamount to withholding the source code, and so is not permitted. Free Software Foundation, *Selling Free Software: High or Low Fees, and the GNU GPL*, FSF WEB SITE (June 23, 2000) <http://www.gnu.org/philosophy/selling.html>.

[17] Free Software Foundation, *Selling Software*, FSF WEB SITE (June 23, 2000) <http://www.gnu.org/philosophy/words-to-avoid.html#SellSoftware>.

[18] *Open Software Licenses: Part 1*, 5.9 INTELLECTUAL PROP. STRATEGIST (June 1999).

In other words, though the GPL has no express provisions against charging for software,[19] it reaches this same result through its apparent appetite for derivative works.

As to derivatives, the GPL gives its licensees the "legal permission to copy, distribute and/or modify the software."[20] Of course, to be a licensee, as noted above, a user must abide by the GPL, which means, among other things, that the licensee must open source (i.e., provide access to the source) all code derived from GPL code.[21] The first section of the GPL's "Terms and Conditions" lays the groundwork for defining derivatives of GPL code:

> The "Program", below, refers to any such program or work [carrying the GPL copyright notice], 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…. 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.[22]

In other words, the license (reasonably) extends to all derivative works, but defers final judgment on whether or not a derivative has been created to a case-by-case analysis of how the Program interacts with outside code. This is more problematic than it sounds, however, because the GPL's definition of derivative works and United States copyright law's definition of derivative works seem to be at odds at times. Stallman may be stepping beyond the bounds of law.

In US copyright law, a derivative work is a work based upon one or more preexisting works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version,

---

[19] Even royalties are permitted, it appears. *See* Email from Heather Meeker to Matt Asay of 5/19/00 (opining that there is no problem charging royalties on GPL code) (On file with Matt Asay).
[20] Free Software Foundation, *GNU General Public License, Preamble*, FSF WEB SITE (June 23, 2000) <http://www.gnu.org/copyleft/gpl.html>.
[21] It also means, as stipulated in the Preamble, that: "Any patent must be licensed for everyone's free use or not licensed at all." *See Id.* This prevents users from making GPL code proprietary by patenting it
[22] Free Software Foundation, *GNU General Public License, Terms and Conditions for Copying, Distribution and Modification*, §0, FSF WEB SITE (June 23, 2000) <http://www.gnu.org/copyleft/gpl.html>.

sound recording, art reproduction, abridgment, condensation, or any other form in which a work may be recast, transformed, or adapted. A work consisting of editorial revisions, annotations, elaborations, or other modifications that, as a whole, represent an original work of authorship, is also a derivative work.[23] Lastly, the copyright owner of an original work is the only one with the right to make or license derivative works based upon a previously copyrighted work.[24]

US copyright law also holds that a work will only be considered an infringing derivative work if it is substantially similar to the copyrighted work, and if the copyright holder did not authorize the derivative.[25] Courts generally consider a work a derivative of a preexisting work if it contains a substantial amount of material from a preexisting work.[26] The courts have not, however, been absolutely clear on whether the linking of software programs, a thorny issue, creates a derivative that the GPL attempts to decide in its favor. This will be discussed in more detail below.

As the GPL stands, it does not advance a precise definition of a derivative of GPL code, precisely because its definition of software linking seems to exceed that found in US common law. Green and Meeker speculate: "The distinction between an add-on (i.e., a new work) and a modification (i.e., a derivative work) in the software world is not always clear-cut. The description of 'a work containing the Program or a portion of it, either verbatim or with modifications' could include a broad mix of modifications and new code."[27] The FSF has indicated that it believes "the details of

---

[23] 17 U.S.C.A. § 101 <http://www4.law.cornell.edu/uscode/unframed/17/101.html>.
[24] 17 U.S.C.A. § 106 <http://www4.law.cornell.edu/uscode/unframed/17/106.html>.
[25] 17 U.S.C.A. § 102 <http://www4.law.cornell.edu/uscode/unframed/17/102.html>.
[26] *See Litchfield v. Spielberg*, 736 F.2d 1352 (9th Cir. 1984) (holding that a work is not derivative unless it has been substantially copied from a previous work, and not merely "vaguely connected" or based on a preexisting work); *Apple Computer, Inc. v. Microsoft Corp.,* 779 F. Supp. 133, 20 U.S.P.Q.2d (BNA) 1236 (N.D. Cal. 1991), related reference, 1992 Copr. L. Dec. P 26903, 1992 WL 75423 (N.D. Cal. 1992) and aff'd, 35 F.3d 1435, 32 U.S.P.Q.2d (BNA) 1086 (9th Cir. 1994).

[27] *Open Software Licenses: Part 2,* 5.10 INTELLECTUAL PROP. STRATEGIST (July 1999).

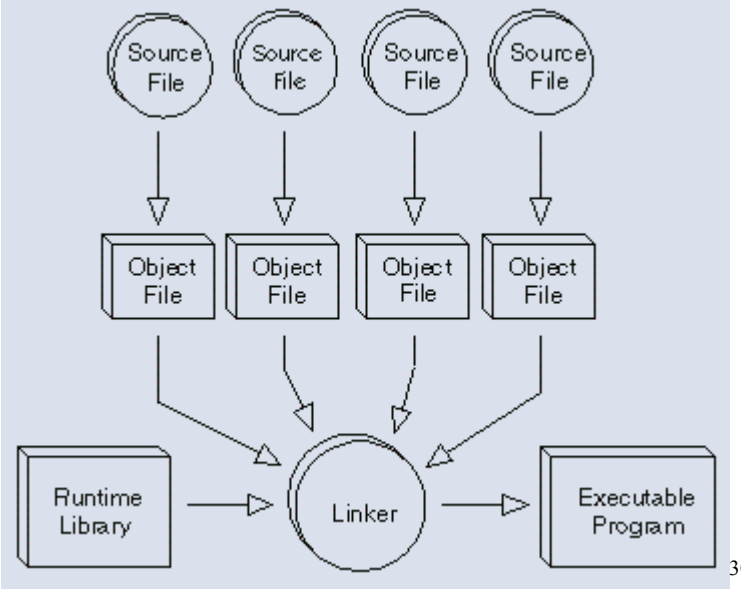
the interaction between the parts determine" whether the program is a modification/derivative.[28] While such a noncommittal stance on "interaction" (i.e., linking) is consistent with US copyright law, the GPL provides further detail deeper in its text, when specifying that its requirements of providing source code and limitations on warranty

> …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 one distribute them as separate works. But when one 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 one; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program. [29]

But what does this mean?

Some authorities suggest that the nature of "linking" between the GPL code and outside code dictates whether a derivative work is created. Linking brings together different modules (and submodules) from different files that are designed to work together, and merges them into one executable program, as depicted in the figure below.

[28] Email from Bradley M. Kuhn, Assistant to Richard M. Stallman, Free Software Foundation, to Matt Asay of 5/19/00 (On file with Matt Asay). *See also* Green and Meeker, *supra* note 24.
[29] Free Software Foundation, GNU General Public License, Terms and Conditions for Copying, Distribution and Modification, §2(c), FSF Web Site (June 23, 2000) <http://www.gnu.org/copyleft/gpl.html>.



[30]

Static linking is the original method used to combine an application program with the parts of various library routines it uses. The linker is given the user's compiled code, containing many unresolved references to library routines. It also gets archive libraries containing each library routine as a separate module. The linker keeps working until there are no more unresolved references and writes out a single file that combines the user's code and a jumbled mixture of modules containing parts of several libraries. The library routines make system calls directly, so a statically linked application is built to work with the kernel's system call interface. The crucial thing to remember is that a static link outputs a combined file from the compiled code and the libraries. This executable would almost certainly be considered a derivative work of the GPL code.

Dynamic linking, on the other hand, does not copy the code into an executable. When the linker builds a dynamically linked application, it resolves all the references to library routines without copying the code into the executable. The status of dynamic linking – whether dynamically linked code creates a derivative – is therefore debatable, at least in the wider software world.

[30] Webopedia, *Link*, INTERNET.COM (Jan. 10, 2001) <http://webopedia.internet.com/TERM/l/link.html>.

It remains to be seen, however, whether the GPL agrees that this issue is debatable.[31] After all, the FSF believes that "there is fundamentally no legal difference between dynamic and static linking."[32] When I asked where they got the idea that there is no "legal difference" between the two forms of linking, they responded that their lawyers told them this was the case, and that they do not know of case law to support (or refute) their position.[33]

The Lesser GPL, or LGPL, is a second software license written by the Free Software Foundation. It sheds some light on the FSF's view of linking and derivative works, both for the GPL and LGPL:

> When a program is linked with a library, whether statically or using a shared library [i.e., dynamically], the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.[34]

In other words, dynamic linking is permissible under the LGPL, but not under the GPL. Whether one links to GPL code dynamically or statically, one creates a derivative work.[35] As Lisa Green and

---

[31] This question is of particular importance to embedded systems. "Embedded" systems include pretty much every computer that is not a server, desktop, or laptop computer. Examples include Palm Pilots, microwaves, garage door openers, and the computers that run automobiles. Many key applications in embedded systems (industrial automation applications, for example) require real-time computing, in which "all software ends up linked into a single executable." In other words, the line between static and dynamic linking becomes blurred in embedded systems. *See* ROSENBERG, *supra* note 10, at 42.

[32] Email from Bradley M. Kuhn, Assistant to Richard M. Stallman, Free Software Foundation, to Matt Asay of 5/20/00 (On file with Matt Asay).

[33] Email from Bradley M. Kuhn, Assistant to Richard M. Stallman, Free Software Foundation, to Matt Asay of 6/26/00 (On file with Matt Asay).

[34] Free Software Foundation, *GNU Lesser General Public License, Preamble*, FSF WEB SITE (June 23, 2000) <http://www.gnu.org/copyleft/lesser.html>.

[35] Green and Meeker concur, positing that:

> [T]he proper distinction [between an add-on/new work and a modification/derivative] is whether a new piece of code is linked to the original code….If the modifications are so integral to the code that no linking is required, the new work is part of the Program. Thus, the line of demarcation may be a technical one – turning on whether linking makes sense for speedy execution.

Heather Meeker point out, software that contains small portions of a copyrighted work that are functional in nature (e.g., a "do loop" that efficiently can only be written in one way) is generally not considered to be a derivative work, but might be according to the GPL.[36]

The GPL's language, then, tries to swallow every piece of software that incorporates a GPL program or portions of it. However, it probably cannot do so under the copyright law it purports to follow. This definition of a derivative work proves highly controversial in the software world, and may not stand up in court.[37] Regardless, however, it has spawned fear in the closed-source world of software development, for fear that linking to GPL code in any manner will cause the closed-source code to be deemed a derivative, and hence subject to the GPL's "open source" requirement.[38] This fear has inhibited scores of computer companies from embracing Linux.

This fear has been caused in large part by the Free Software Foundation's inability or unwillingness to clarify its position on derivative works. I asked two prominent representatives of the Free Software Foundation – Eben Moglen, general counsel, and Richard Stallman, founder – to clarify thorny issues of linkage to GPL code, and came up with two divergent opinions on derivative works in specific contexts. Their responses (to the question of whether or not they would consider the following derivative works) are recorded below:

Lisa Green and Heather Meeker, *Open Software Licenses: Part 2*, 5.10 INTELLECTUAL PROP. STRATEGIST (July 1999).

[36] *Id*.

[37] To help clarify the meaning of the GPL, Matt Harris, CEO of embedded Linux software company, Lineo, has proposed Revision 3.0 of the GPL, as found in the attached Appendix A.

[38] The fear is well placed, for several reasons. One prominent reason stems from the consequence of being wrong. If code is found to be derived from the GPL, and the company will not open source their software, then the company loses, not only their copyrights, but the legal permission to use the original GPL code, as well (which is tantamount to forfeiting one's copyrights). *See SAS Institute, Inc. v. S & H Computer Systems, Inc.*, 605 F. Supp. 816 (M.D. Tenn. 1985) (holding that where a licensee of a computer program adapted the program for purposes not allowed under the licensing agreement, an infringing derivative work was produced, and the copyright was therefore invalid).

A driver loaded as a module into the Linux kernel?

> **Moglen**: No
>
> **Stallman**: Yes. I think Linus made a mistake when he said he would interpret the GPL so as to regard these as NOT extensions to Linux.

A module written to be plugged into an API defined specifically to support dynamic loading?

> **Moglen**: No.
>
> **Stallman**: It depends on the detailed circumstances.

A program which uses a library? (i.e., Would it be fair to say that this is generally not a derivative work of that library?)

> **Moglen**: This depends. Code statically linked to other code is a derivative work of the code with which it is linked, as far as we are concerned. But what follows from that depends on the license involved. That's the difference between the GPL and the LGPL, which is designed to permit proprietary code to be linked and distributed with free libraries.
>
> **Stallman**: I think that depends on the detailed circumstances.

A library linked to a program? (i.e., Is this a derivative work of the program?)

> **Moglen**: Code statically linked to code constitutes a derivative work of the code to which it is linked, without question, regardless of license terms. More specifically, now regarding licensing as well as the status of the work, code that cannot be used at all unless dynamically linked to GPL'd code, and which is distributed along with that GPL'd code, must be distributed under the terms of the GPL. This provides a competitive advantage to free software, requiring those who wish to make unfree software to undertake proprietary reimplementation of feature sets only available in GPL'd libraries, such as GNU readline.
>
> **Stallman**: That is normally true, but it one needs to be careful drawing conclusions from it. Also, there can be ambiguity in the definition of "library" which can cause confusion about this.

A program running as a process on a Linux system? (A derivative work of the Linux kernel?)

> **Moglen**: Certainly not a derivative work of the kernel, or of anything else, simply by virtue of executability on free software systems,

> whether the kernel they employ is Linux, the Hurd, or some other kernel.
>
> **Stallman**: I agree here [that the program running as a process on a Linux system creates a derivative work]--except that it's a misnomer to speak of "a Linux system". Linux is the kernel; the system is really the GNU operating system, modified to use Linux as the kernel.[39]

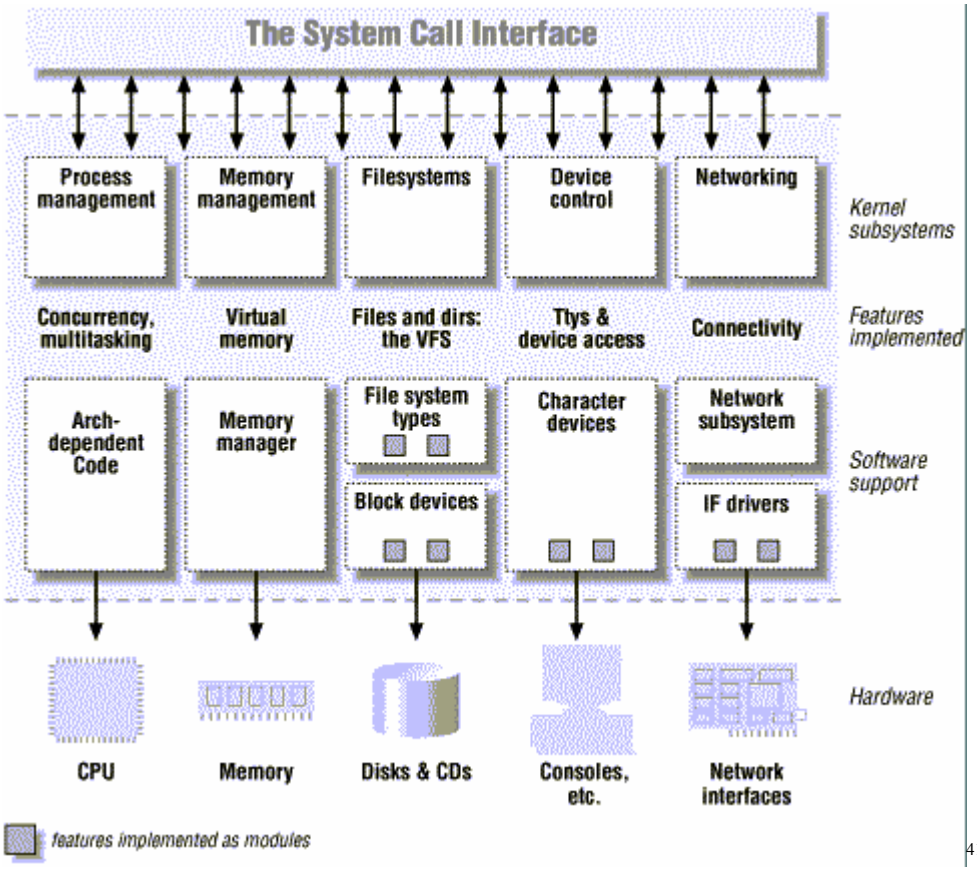
To be fair, my questions came without context or clarification, so a certain amount of divergence could be expected. Still, it is telling how widely their responses diverge – there appear to be no definitive answers to the question of what constitutes a derivative work under the GPL, not even from the holders of the licenses in question. This uncertainty may well be the most nettlesome issue inhibiting the widespread adoption of Linux by computer makers.

Fortunately, as I will detail below, this issue has largely gone away, as it has become accepted practice to dynamically link to GPL code. Linus Torvalds helped to build momentum for such a reading of the GPL. While some argue that kernel modules, including device drivers, must be GPL, Torvalds has stated: "This [GPL] copyright does *not* cover user programs that use kernel services by normal system calls – this is merely considered normal use of the kernel, and does *not* fall under the heading of 'derived work.'"[40] Hence, any applications that run in "user space," (e.g., Word processing program, email program, etc.), or those that interact with the kernel in a standard way, common to programmers, is acceptable. In turn, this reading leaves a great deal of software room to interact with the GPL'd kernel, without being consumed by it, as shown in this diagram. (The features implemented as modules should be free of GPL contamination.)

---

[39] Email question-and-answer "interview" with Eben Moglen, General Counsel of the Free Software Foundation and Professor of Law at Columbia Law School, and Richard M. Stallman, Founder of the Free Software Foundation (May 26, 2000, and May 27, 2000, respectively) (email on file with author).

[40] From the COPYING file in Linux. This file contains the text of the GNU General Public License, but it begins with a "NOTE!" which contains the sentence above.



[41]

Giving further legal support to this view, as Jerry Epplin has written, kernel developers have consistently supported such an open interpretation by accepting the presence of proprietary hardware drivers. This tolerance of proprietary drivers has held for a very long time, and numerous hardware manufacturers currently depend on it.[42] This essentially means that the core of the system is off-limits (for tampering without sharing), but that the core is open as an acceptable platform to build upon by creating programs that leverage its functionality without stealing that functionality and calling it one's own. This reading of the GPL has become so prevalent in the past year that Stallman's opinion on the matter is largely irrelevant. The GPL has effectively merged with the industry standard reading of derivatives in US software law.

[41] RUBINI, ALESSANDRO AND JONATHAN CORBET, LINUX DEVICE DRIVERS, 2ND ED. (June 2001), found at < http://www.xml.com/ldd/chapter/book/ch01.html>.

[42] *Using GPL Software in Embedded Applications*, LINUXDEVICES.COM (Sept. 30, 2002) <http://www.linuxdevices.com/articles/AT9161119242.html>.

The last important issue arising from the GPL revolves around the source code revelation requirement. Under the terms of Section Three, developers must provide the source code to any derivatives of GPL code. Developers can distribute their software in binary form (i.e., object code), but only if they do the following: "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…."[43] In other words, if one creates a derivative, one must license the derivative under the GPL, and must reveal the source code to one's program to *all* third parties, including one's direct competitors (assuming the creator of the derivative work has competitors).

In sum, users of the GPL code are empowered to do pretty much whatever they want with the GPL code, provided that they assert no proprietary rights to the original code and open source any derivative works. Untangling the question of what constitutes a derivative work is the thorniest issue the GPL raises, and was by far the biggest roadblock to Linux adoption by the world. However, as noted, over the past year this roadblock has effectively dissipated, as legal uncertainty gave way to business practicality.

As I mentioned at the start of this paper, I have been deeply embroiled in the Linux movement for the past two years. Two years ago, just about every electronics company had someone appointed to figure out Linux, but almost no one was buying. A year ago, companies were deeply interested in moving to Linux, but were still worried by the GPL, and Lineo (my employer), a legal innovator of sorts, was pushed to indemnify all customers against GPL contamination and to build a software tool that would scan a customer's code to search for GPL infractions, and suggest alternative development

[43] Free Software Foundation, *GNU General Public License, Terms and Conditions for Copying, Distribution and Modification*, §3(a)(b), FSF WEB SITE (June 23, 2000) <http://www.gnu.org/copyleft/gpl.html>.

methodologies to get around the problem. Now, it appears that the dam has burst, and the GPL is largely an afterthought. The concern that crippled the move to Linux a year ago has almost completely evaporated. Companies have become savvy to what the GPL does and does not mean, and the Free Software Foundation has opened up to most uses of dynamic linking to GPL code.

Of course, in an important sense, all software licenses are viral in nature, so Linux should not have posed more of a threat than, say, Microsoft's Windows operating system. Any copyright holder has the right to restrict and control derivative works, and the original copyright follows the licensee to whatever licensed derivatives they create. The GPL, it can be argued, is little different from standard software contracts. Charles C. Mann makes this point effectively (and humorously) in an email communication he sent me. In responding to my claim that the GPL stifles innovation through its restrictive license terms, Mann replied: "Copyright owners have always had the ability to refuse to license their material unless the licensees jump through various hoops – pay a lot of money, use it only in certain ways, push enchiladas down 5th Avenue with their noses, etc."[44]

The primary difference, the argument continues, is that whereas the "Microsofts" of the world make unreasonable demands and personally profit from doing so (in a monetary sense), Stallman and his GPL make unreasonable demands but derive no pecuniary profit from doing so. Instead, Stallman is simply content to see his free software cause advanced with every addition to the GPL's portfolio of licensed code.

This is all true, to a point. Copyright and copyleft licenses prove similar in the end result: creating a derivative of Microsoft-licensed software is illegal and will get one sued; creating a

[44] Email communication from Charles C. Mann, contributor to *The Atlantic Monthly* (April 27, 2001) (email on file with Matt Asay).

derivative of GPL-licensed software is legal but will get one sued if one subsequently fails to open source ones' software. Both restrict derivatives, and punish deviation from their terms.

The difference, however, is that Microsoft and the closed-source software licensors of the world generally encourage controlled linking to their code in an effort to build up network effects, thereby increasing the general utility of their software. Dynamic linking is the norm, and does not generally trigger lawsuits under US copyright law. In the case of the GPL and free source, on the other hand, the network effect is (or, rather, was) forced, rather than encouraged, by defining all linking as creating derivatives. This definition is much more restrictive than standard industry practice, and ends up closing off the free-source world to closed-source companies.

Until the sea change, then, it did not really matter if the GPL's view of derivatives was correct. No one wanted to take the chance. The consequences of getting the answer wrong – potentially having to open source the offending code, or losing one's copyright protection to the code – were too dire. Companies who simply wanted to build applications and drivers to extend the functionality of Linux for their customers were prevented therefore from doing so by the voracious GPL appetite for all things linkable, or at least the specter of such. Whichever it was, the result was the same, because perception is, to a great extent, reality. By closing off this avenue of software development to GPL code (like Linux), or at least fostering the appearance of doing so, the GPL road-blocked the consumer welfare that would have resulted from consumers' access to Linux' robust, stable code.

So, in a real sense, my prior position was correct. To the extent that the GPL interfered with US copyright law to disallow normal uses of dynamic linking, it was an innovation-killer. I believe, in retrospect, that the GPL's anti-linking stance derived more from the myths about what

the GPL meant, rather than from any reality. Now that the myths have largely been forgotten, and companies and individuals feel comfortable working with Linux, the GPL is poised to promote innovation, as I will detail below.

The Economics of the GPL: Is This a Good Thing?[45]

Old View: GPL Kills Innovation

In my previous paper, I argued that regardless of whether the GPL and its definition of derivatives are legally enforceable, the license is fraught with a much more fundamental problem: economic viability. I want to present that argument here, in its best light, and then work on quashing it in the next section. Once the misinformation is stripped away from the GPL, there is a strong argument to be made that open-source software, and open code generally, will be a primary engine to drive innovation in software.

But first, my former (and wrong) view. As I noted above, Linux has found its way into servers, where the users of the code are qualified software engineers. Linux has not, on the other hand, made much of a dent (Two percent, according to IDC) to date in the consumer world. That is, in actually finding its way onto devices that ship to consumers. I argued that the GPL, by eliminating property rights in software, and by failing to clarify the rights users *do* have, has made Linux a nonentity to the majority of the world.

This caused me serious concern, given Linux's generally recognized technical superiority to Windows and Macintosh, and hence its ability to benefit the computing world. The GPL keeps Linux out of the hands of consumers, all but forcing them to use Microsoft products, at a much higher price, and with much less utility. In other words, the FSF uses copyright law to

[45] For a good overview of the economic considerations affecting open-source software, see Stephen M. McJohn, *The Paradoxes of Free Software*, GEORGE MASON L.R., 25, 36-45 (Fall 2000).

achieve the very opposite of copyright's constitutionally stated purpose, which is to promote science and the useful arts by securing a limited monopoly to authors. Such monopoly rent provides the author a property right in their work, with the attendant incentive (and ability) to exploit the creation.

The GPL turns all of this on its head. Not only does the GPL remove the property right (or, at least, the ability to profit from it),[46] the GPL skews the definition of "common good" for which copyright and patents provide. That is, the GPL focuses on developers, rather than the end user. Donald Rosenberg discusses this tendency in the free software mentality:

> …[Stallman] sees the user [of GPL software] as a competent software developer, not a point-and-click end-user. This developer-centric attitude ignores the vast body of users who are happy with binary-only software, and it defines the only way of making money from Free Software… as developer-provided services: program modification or extension, technical support, technical documentation, and training. An end-user who cannot program can hire a developer to make changes in a program, provided the source code and permission to do so are available. This is a hacker-centric view of the world, one that defines computer literacy as programming facility.[47]

Stallman has stated on several occasions that his vision is developer-centric, and that he cares very little if GPL code is ever "popular" (with the apparent meaning being that it is used by a large number of average people). This type of thinking overlooks *total welfare* that, as Landes and Posner argue, "depends on the number of works created as well as on the consumer and

---

[46] William M. Landes and Richard A. Posner explain:

> In [the absence of copyright protection] anyone can buy a copy of [a work] when it first appears and make and sell copies of it. The market price of the [work] will eventually be bid down to the marginal cost of copying, with the unfortunate result being that the [work] will not be produced in the first place, because the author and the publisher will not be able to recover their costs of creating the work. The problem is magnified by the fact that the author's cost of creating the work…[is] incurred before it is known what the demand for the work will be.

William M. Landes and Richard A. Posner, *An Economic Analysis of Copyright Law*, XVIII J. LEGAL STUDIES 325, 328 (1989).

[47] ROSENBERG, *supra* note 11, at 19.

producer surplus generated by a given work, assuming it is created."[48] The GPL fails to maximize total welfare to the extent that it inhibits the creation of works and drains the consumer and producer surplus.

GPL and Innovation[49]

The GPL thereby expressly destroys the whole intent behind the copyright system. By effectively disallowing an author's/coder's return on investment, the GPL limits the amount of code written. The GPL exacerbates this by extending itself to products designed to interoperate with it, which would normally be acceptable under copyright law. In this way, it tries to embrace and extend copyright, and fends off would-be creators of applications and middleware for GPL code, thereby stifling innovation and reducing the general public's access to this solid code.

In consequence, GPL code development is left to a volunteer army of dedicated hackers.[50] While this army has managed to turn Linux into a significant operating system, they have not managed to get out in front of development. Free source software has a long history of playing catch-up to proprietary software. Some open-source proponents like Pavlicek dispute this,[51] but they have difficulty articulating an argument that overcomes Linux' history of trying

[48] Landes and Posner, *supra* note 44, at 341.
[49] For a contrary view, one that holds that copyright has done innovation in software a grave disservice, see Mark Haynes, *Black Holes of Innovation in the Software Arts*, 14 BERK. TECH. L. J. 503(1999). *But see* Mark A. Lemley and David McGowan, *Legal Implications of Network Economic Effects*, 86 CAL. L.R., 479-610 (1998); Peter S. Menell, *Tailoring Legal Protection for Computer Software*, 41 STAN. L.R., 1329-1372 (1987); Peter S. Menell, *An Analysis of the Scope of Copyright Protection for Application Programs*, 41 STAN. L.R., 1045-1104 (1989).
[50] In the open-source software world, "hackers" is a title of distinction and respect – it refers to any qualified software engineer. "Cracker" is the label applied by the open-source world to the software pirates that the non-software world normally ascribes to the word "hacker."
[51] *See* source cited *infra* note 2, at 64, 96.

to mimic the best in UNIX,[52] and now Windows (Eazel, gimp, etc.). The number of significant exceptions can be counted on one hand.

Intellectual property law must balance between incentives to the producer of information and the consumers of such information. Where there is too much protection, producers will collect overly high (and, hence, inefficient) rents. Because producers are also consumers (in that they compile a work from existing copyrighted works), they also lose out from overbroad IP protection. On the other hand, too little protection gives producers no incentive to produce, thereby creating a provisioning problem for end users. William M. Landes and Richard A. Posner, noted authorities in the field of Law and Economics, explain:

> Copyright protection – the right of the copyright's owner to prevent others from making copies – trades off the costs of limiting access to a work against the benefits of providing incentives to create the work in the first place. Striking the correct balance between access and incentives is the central problem of copyright law. For copyright law to promote economic efficiency, its principal legal doctrines must, at least approximately, maximize the benefits from creating additional works minus both the losses from limiting access and the costs of administering copyright protection.[53]

Creators generally need some form of return on their investment of time and money in order to produce for others. Whereas it may be true that closed-source licensing provides too much protection, it is equally likely that the GPL provides too little. The GPL is an incentive killer, just as much as Stallman would argue Microsoft-type licensing is a freedom-killer. The best alternative must be somewhere between the two extremes.

Eric S. Raymond, a luminary in the Open Source world, argues that open-source software skirts this problem, for the most part, because of the different incentives driving open source. Raymond claims, "The "utility function" Linux hackers are maximizing is not classically

[52] Torvalds, himself, acknowledged in the early days of development on the Linux kernel that much of the work on Linux was simply an effort to clone UNIX. *See* PETER WAYNER, FREE FOR ALL: HOW LINUX AND THE FREE SOFTWARE MOVEMENT UNDERCUT THE HIGH-TECH TITANS 62 (2000).
[53] Landes and Posner, *supra* note 44, at 325, 326.

economic, but is the tangible reward of their own ego satisfaction and reputation among other hackers."[54] Raymond thus builds a pseudo-economic model built upon the currency of reputation, rather than money, arguing that hackers will form markets in reputation capital, which in turn will advance open-source software development.[55]

Unfortunately, such reasoning, however valid it may be within the open-source world, fails to address the much larger remainder of the development "pie." That is, it does not address the ways in which open source fails to innovate, create applications and other software, and bring outsiders into the open-source development effort. As a result, the average consumer is still locked out of the benefits of open-source software like Linux, and will almost certainly remain so until Stallman opens up the GPL to open licensing schemes.

The GPL and Applications

Very few applications exist in the open-source world, at least those that would be useful to the average user. Why are there so few applications for Linux? The most obvious answer is that the GPL diminishes the incentive to develop Linux code, since that code must immediately be given away to one's competitors. It is also true that Linux has no network effect (yet) to leverage, so corporate developers prefer to spend their money on projects closer to large, established markets. But more fundamentally, the GPL squelches any movement toward

[54] ERIC S. RAYMOND, *The Cathedral and the Bazaar*, *in* THE CATHEDRAL AND THE BAZAAR: MUSINGS ON LINUX AND OPEN SOURCE BY AN ACCIDENTAL REVOLUTIONARY 27, 64 (1999).

[55] Lerner and Tirole advance a slightly more complex, and likely accurate, argument as to the open-source incentive structure. They posit that there is a signaling incentive involved in open source, whereby developers advance their careers by advancing their reputation in the open-source community. Reputation is thus a means to both an intangible end (Stature amongst one's peers), as well as a tangible end (Better job prospects, at greater pay). *Supra* note 4, at 15, 17.

building the requisite "network,"[56] by insisting on a view of copyright law that disallows (or strongly implies such) dynamic linking to GPL code, without in turn open sourcing one's code.

Given that the GPL thwarts closed-source developers' attempts to work with Linux, why has the Free and Open Source world failed to fill the application void? One reason is that coders may not have this "itch," and so it never gets "scratched."[57] Brian Behlendorf, co-founder of the Apache Web Server Project, provides several other reasons for open-source software's traditional focus on the infrastrutural/back-end side of software:

- End-user applications are hard to write, not only because a programmer has to deal with a graphical, windowed environment which is constantly changing, nonstandard, and buggy simply because of its complexity, but also because most programmers are not good graphical interface designers, with notable exceptions.
- Culturally, open-source software has been conducted in the networking code and operating system space for years.
- Open-source tends to thrive where incremental change is rewarded, and historically that has meant back-end systems more than front-ends.
- Much open-source software was written by engineers to solve a task they had to do while developing commercial software or services; so the primary audience was…other engineers.[58]

In other words, open-source developers have neither the history of developing applications, nor the future of doing so, given that such expertise is not generally found within the open-source ranks. This is so for several reasons, but one (unspoken) reason is that applications are viewed within the software world as "wimpy." Real men (and most developers *are*, in fact, men) develop the low-level networking layers of code; first-year-out-of-an-undergraduate-degree developers write application code.

---

[56] *But see* PAVLICEK, *supra* note 2, at 96 (arguing that the revolutionary software advances of the century have developed under open source, rather than proprietary, models).

[57] This analogy comes from Eric S. Raymond's classic Open Source defense, *The Cathedral and the Bazaar*. In it Raymond argues that Open Source development proceeds by a series of itches (i.e., Needs that coders must solve) and scratches (The software solutions to those itches). RAYMOND, *supra* note 52, at 32.

[58] Brian Behlendorf, *Open Source as a Business Strategy*, *in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION 149, 159 (1999).

More fundamentally, consumer products require an immense amount of unsexy code – open-source developers do not necessarily want to invest their time on such parts of the code base, preferring to spend time on more visible, reputation-enhancing projects.[59] Drivers, compatibility code, etc. make up the bulk of a complete software program, but they are the least interesting to write. Open Source must rely on a volunteer army. With no authority figures beyond the package manager for a given bit of code, no one wants to be a private in this army. People must be paid to do such "grunt" work. Corporations are able to employ people and direct them to write the tedious bits of code (which, by some accounts, make up as much as 90 percent of any program).

As a result, the GPL may be marginalizing Linux as Microsoft takes the world to a new computing model, .Net, because Linux' volunteer network will not be able to keep up with Microsoft's R&D and marketing machines.[60] Craig Mundie, a senior executive with Microsoft, spoke to this point in response to a question:

> Question: According to several participants on a Linux panel at Comdex, Linux will usurp Windows as the key operating system in the Internet era, especially overseas. What's your response?
>
> Mundie's Answer: I think to some extent it's going to be hard to find how resources in the open source environment are going to get mustered to develop tools for next-generation services. If people are satisfied living in the browser model and they want to buy free software, then Linux will be a choice. But we

[59] Eric Raymond attempts to rebut this argument by enshrining its inverse as a "rule" of software: "Continued devotion to hard, boring work (like debugging, or writing documentation) is more praiseworthy than cherrypicking the fun and easy hacks." *Homesteading the Noosphere*, *in* THE CATHEDRAL AND THE BAZAAR: MUSINGS ON LINUX AND OPEN SOURCE BY AN ACCIDENTAL REVOLUTIONARY 79, 117 (1999). Unfortunately, calling something a rule does not make it such, and few developers would agree that the "dirt work" of software development is as rewarding, in any way, as developing a networking protocol, or some such feature, and would subsequently invest their time in the "dirt work." *See also* PAVLICEK, *supra* note 2, at 56-57 (proposing that mundane tasks will be interesting to someone, since every developer will enjoy different tasks, and that developers are professionals who know that mundane tasks must be done, whether they are "sexy" or not).

[60] This is highly unfortunate, given Linux' superiority in terms of networking capabilities. Linux should be the Microsoft-killer in a .Net world that relies on networked, server-based computing. *See, e.g.,* ROSENBERG, *supra* note 11, at 201.

> don't think that's going to be a long-term model that predominates in the Internet era.[61]

Spin aside, Mundie is almost certainly correct, if Linux' history is any indication of its future ability to innovate.

Inefficiency of the GPL

In addition to its stunting effect on innovation, the GPL is inefficient. Because the GPL severely restricts the manner in which proprietary code can interact with it, the GPL forces costly designing around it, if one is to attempt to replicate the best in Linux for another OS. This is similar to the design-around problem with patents, but is arguably more egregious, because there is no way to license one's way out of the problem. Also, unlike in patent law, where there are well-established rules for allowing advances on patented code, Stallman and the GPL proclaim the unavailability of such routes to Free Source developers.

Thus, closed-source development must redevelop GPL code in clean room-type environments, without the ability to effectively shield itself from exposure to the code, because the code is everywhere available. This makes it difficult to impossible to persuade a court that the developers had no access to the GPL code during development. Regardless, no one is about to redesign Linux, in a clean room or otherwise. Stallman's GPL proclaims that participants in Linux development must be "free sourcers," or have no rights to the software. This hard-line stance keeps the bulk of the world from benefiting from Linux.

Lastly, in an effort to escape the restrictions of the GPL, many companies are starting to consider programming fixes such as CORBA[62] and other programming techniques to buffer

---

[61] Paul Rooney, *A Chat with Craig Mundie*, TECHWEB (Nov 17, 2000) <http://www.techweb.com/wire/story/TWB20001117S0001>. *But see* Jim Farley, *Microsoft .Net vs. J2EE: How Do They Stack Up?*, O'REILLY.COM (Aug. 2000) < http://java.oreilly.com/news/farley_0800.html>.

closed-source code from GPL code, thereby avoiding dynamic or static linking to GPL code. Such development introduces inefficiencies into the code (Causing the code to function slower, or take up more memory), and adds unnecessary cost to development.

Again, Mann would counter that this is true of all copyrights/patents. However, in every other form of intellectual property, there is at least one group/individual that benefits (profits) from the IP protection. In GPL software, no one does, in a monetary sense, except those companies that have created service-based business models around Linux. Such companies, however, have yet to prove their ability to scale to meet the demands of the software market, and there is every reason to believe that they will continue to fail in this (given their revenues and profit margins).

Also, and more fundamentally, in GPL code there is no fair use defense (at least as defined by the GPL) to allow reasonable copying of the code. The would-be copyist is therefore forced to either concede to the GPL, or abandon the project – either route is inefficient for the end consumer (as it devours larger and larger shares of the consumer surplus). Because of these abnormal restrictions, many would-be Linux developers abandon development, making proprietary code (and GPL code) that much less useful. The less it Linux used, the less benefit the consumer derives from it

This is the greatest argument against the GPL: It actually hurts the consumer good. This is not a surprising result, given that the GPL's focus is wholly on producers, and not consumers. The GPL fails to see the world beyond the developer community. Whereas the Constitution calls for protection of producers to enhance the public good, the GPL calls for protection of producers

---

[62] *See* Michael Trachtman, *Can Corba Sidestep Open-Source Licensing?*, WEBTECHNIQUES, April 2001, at 48. *See also* ROSENBERG, *supra* note 9, at 95.

(and only one kind of producer – the "talented tenth" of open-source software developers) to enhance the producers' good.[63]

Windows managers (like Eazel) may make life better for the average consumer, because they make Linux more intuitive for the average user. However, beyond the question of whether such companies will be able to financially survive,[64] given the fact that they are effectively foreclosed from profiting from their innovation, there is the wider question of whether consumers will care about a pretty interface if few useful applications exist for Linux.

In this way, Stallman is little different from Bill Gates: Stallman, too, seeks monopoly to further his own, narrow community's interests. For Gates, it is the shareholders and himself. For Stallman, it is the free source developers and himself. By attempting to force the world to conform to his own, narrow vision of ethics and morality, Stallman has created a niche in the world that benefits only developers of open-source code, with the remainder of the world stuck with buggy, unstable Windows.

New View: GPL Promotes Innovation

Over the past year, I have found my former view of the GPL slipping into strong support and admiration. Part of this came as the industry (and I) grew comfortable with what the GPL meant, or what it would allow. But part of it came as I watched Microsoft continue barreling forward into new markets with little but monopoly power to recommend it. Microsoft writes

[63] "Linux," Torvalds warned early on, "is a program for hackers by a hacker." Indeed, UNIX, the inspiration for GNU and Linux, is exactly the kind of software that makes the uninitiated feel stupid. Militantly unintuitive, it requires the mastering of dozens of telegraphic commands such as "egrep," "ping," and "gzip," all of which can be modified to act in any number of ways familiar to anyone who remembers MS-DOS. (Charles C. Mann, *Living with Linux (Part 2)*, THE ATLANTIC ONLINE (Aug. 1999) <http://www.theatlantic.com/issues/99aug/9908linux2.htm>.

[64] Unlikely, especially in Eazel's case. One of the more successful attempts to popularize Linux for consumers, the company announced on May 17, 2001, that it was forced to close operations.

good code; Microsoft does not write great code. As someone supportive of the idea of a meritocracy, I wanted to see Linux's great code displace Microsoft.

And it has, for several reasons. For one thing, the GPL has provided a proprietary-free zone in which developers can collaboratively develop code without fearing that they will not be fairly compensated for the work. Because their up-front expected compensation is zero (except in terms of reputation capital or in seeing a problem solved), open-source developers appear to be content to receive no remuneration for their work…provided no one else does, either.

More importantly, the GPL provides for fertile innovative ground where it is most necessary: the foundational or operating system level. Not only does the GPL ensure a robust platform upon which to build "killer apps," but the GPL also frees up resources to work on such apps. Individuals and corporations are enabled to free ride on the platform development (which continues to barrel forth, unfazed by this alleged "overgrazing"), leaving their financial resources available to work on building the superstructure. In this way, the GPL is extraordinarily efficient, pushing innovation out to the edge of the network, as it were, leaving the platform open to development by those with no ulterior motive beyond excellent code.

GPL and Innovation, Revised

A year ago, I chided open-source development as an exercise in spiteful innovation-blocking. It has become clear since then that the GPL has fostered, not fractured, innovation in Linux (and other code). This innovation has come because, and not in spite of, the GPL's insistence that code be free. As open ground has been cleared where innovation can happen for innovation's sake, a robust platform has emerged upon which to build either open or closed-source applications.

One example of GPL-inspired innovation is Linux's superior networking features. It is an operating system that was designed from inception for networked devices, unlike Microsoft Windows and Wind River Systems' VxWorks (the dominant embedded operating system), which have patched in such capability, with less-than-ideal results. Microsoft, for example, recently trumpeted its development of IPv6[65] functionality into the Windows operating systems.[66] In reply, the Linux world let out a deafening yawn: Linux has had IPv6 support for several years (and was the first operating system to have such support). IPv6 support, however, is but one example of Linux's superiority. The current Linux kernel version 2.4 has a large menu of features in it – all of them "standard" – which other operating systems (like VxWorks) have difficulty offering even for a substantial fee. Then there is the modularity of Linux, which allows different modules to be pieced together like a puzzle, with excellent interoperability without much tinkering with the code. This may not seem like a particularly important innovation to the lay reader; to the programmer, however, it is nirvana.

Other examples of how the GPL has fostered innovation move beyond the operating system. The Mozilla browser, for example, actually remembers a user's preference to not accept cookies from a particular site. (Most browsers refuse to "believe" the user's intent, and continually pass through cookies to the user. The audacity of Mozilla to actually care about user preferences!) This consumer-friendly feature is something one simply does not find in commercially available browsers, because there is no financial incentive to block cookies. In

[65] IPv6 is a next-generation addressing system that overcomes the IP address shortage found in the current system, IPv4. It is a highly significant, and very difficult, technological accomplishment. *See* IPv6.org, IPv6: Networking for the 21st Century, IPV6.ORG WEB SITE (April 2, 2002) <http://www.ipv6.org/>.

[66] *See* Microsoft, Microsoft Research IPv6 Implementation, MICROSOFT.COM WEB SITE (April 2, 2002) <http://research.microsoft.com/msripv6/>.

fact, quite the reverse. Only open-source software, provided as it is without pecuniary incentive, enables consumers to actually write code for other consumers.[67]

It may be true, as some (including I) have argued, that open-source development generally plays catch-up to proprietary software. At the application level, that is almost certainly true, most of the time. But I am less convinced that this is a damning argument than I once was. So long as innovation at the core OS layer is meeting or exceeding the innovations of proprietary operating systems (and it is), and so long as open-source software applications can maintain a short market entry lag-time behind their proprietary cousins, with superior code once they do "catch up," then this is no longer a concern. At any rate, once the Linux platform truly catches on (as it is now), the application lag will fade.

As a case in point, there are currently no Open Source versions of UPnP, an architecture for pervasive peer-to-peer networking (such that my UPnP-enabled toaster, for example, will immediately recognize itself and all other UPnP-enabled devices on a network, and just as immediately be able to interact with them). Microsoft developed this technology, and it appears to be a hit in the market. While there are Linux implementations of UPnP, there is not yet an Open Source version of UPnP. But there soon will be, because Intel released its source for Linux-based UPnP to the world in 2001.[68] And when it is, it will come without the royalty schedules that closed-source iterations espouse, and will be equally (or more) robust. *That* is true innovation.

---

[67] Professor Larry Lessig addresses this issue of narrow commercial innovation (to serve narrow commercial interests) in his book, *The Future of Ideas: The Fate of the Commons in a Connected World*." As but one example of how corporations can twist innovation to meet shareholder, but not wider societal values, Lessig cites AT&T's monopoly on telecommunications services. *See* LAWRENCE LESSIG, THE FUTURE OF IDEAS: THE FATE OF THE COMMONS IN A CONNECTED WORLD, 26-48 (2001).

[68] *See* SourceForge, UPnP SDK for Linux, SourceForge Web Site (April 3, 2002) <http://upnp.sourceforge.net/>.

The reasons behind Linux's openness to innovation are not hard to find. Just as Professor Lessig argues for the opening up of radio spectrum to competitive use,[69] so, too, does it make sense to leave the "code layer" open to non-exclusive development. This is what Linux does, and this is one reason that Linux thrives. With no one entity controlling the kernel, Linux can be as much (or as little) as its contributors want it to be. To date, they have wanted it to be equal to and greater than proprietary operating systems like Windows, and they have attained this goal. Doing so has made it an increasingly interesting platform for application developers (at Professor Lessig's "edge of the network," as it were) to build on. And because the GPL permits the dynamic linking of proprietary code into its core, this innovation at the edges includes both open-source and closed-source applications, with increasingly rich effect.

In short, all code need not be closed to induce innovation. In fact, much code must be open if it is to thrive. A mix is necessary. Open-source development can only go so far on its own. At some point, the Toshibas of the world will not espouse an open OS like Linux if they cannot link their intellectual property to it to create a hybrid solution.[70] At the same time, closed-source development hinders itself by closing off promising avenues of development, simply because the corporation lacks the inclination or resources to develop in a particular direction.

Lessig makes this distinction clear. He separates rivalrous from nonrivalrous goods, and argues that different levels of control are needed for the two types of goods.

---

[69] *See* Lessig, *supra* note 65, at 241.

[70] Just as a side note, many closed-source companies are beginning to release their proprietary code to the world. Netscape's Mozilla project is but one (and a poor one) example of this trend. Much more interesting are the unpublicized, but much more numerous, examples of this. Take, for example, Sharp's OpenPDA.net initiative, where they are opening up the specs for their hardware (and software) platforms, encouraging others to develop (and claim) on the platforms to create winning handheld devices. Or like Integrated Device Technology , Arcturus/Samsung, and Sun open-sourcing the board support packages (The software that enables Linux to interact with silicon, e.g., a development board) for the IDT 355, Samsung 4530, and Sun UltraSparc IIe processors. These "small" movements toward open source are highly significant, because they make it immediately possible to develop for winning silicon platforms, without going through months of development effort (on drivers, etc.).

1. If the resource is rivalrous, then a system of control is needed to assure that the resource is not depleted – which means the system must assure the resource is both *produced* and not *overused*.
2. If the resource is nonrivalrous, then a system of control is needed simple to *assure the resource is created* – a provisioning problem….Once it is created, there is no danger the resource will be depleted. By definition, a nonrivalrous resource cannot be used up.

> What follows then is critical. The system of control that we erect for rivalrous resources (land, cars, computers) *is not necessarily appropriate* for nonrivalrous resources (ideas, music, expression). Indeed, the *same system for both kinds of resources may do real harm*….One size won't fit all.[71]

Whatever the legal protections, the GPL has provided for mixed legal protection, and hence a hybrid development model, for Linux. Open and closed source exist side by side in Linux, and innovation has been fast and furious as a result. It is the GPL's insistence that code be cumulative, and not allowed to shutter itself away, which makes it such an innovation lightning rod.

Therefore, my earlier argument about Linux and innovation gets washed away in the face of reality. Linux is thriving. Something has changed in the mood of high technology – there are increasingly only two games in town. You are either developing for Linux (or, less likely, some other open-source variant), or you are developing for Microsoft. Very little middle ground remains. As the GPL continues to provide an open platform upon which to develop, innovation in Linux will continue to outpace that of its closed-source competitors.

[71] *See* LESSIG, *supra* note 65, at 95.

The GPL and Applications, Revised

It may have been true a year ago that there were few winning applications for Linux. It was certainly true that there were few that could not be dismissed as imitative, rather than innovative. That is no longer the case. Today, Linux users have access to the same range of software applications that users of Windows do, though the Linux versions of such software (comparing StarOffice to Microsoft Office, for example) are not yet full competitors.[72] Importantly, as computing increasingly moves off the desktop to server-based and embedded devices (PDAs, smartphones, information appliances, etc.), the consumer's bond with Microsoft and its software fades, and a new battleground is formed, one which Linux (coupled with Java) is better equipped to fight than Microsoft. Life moves beyond Word and Excel in a handheld device, preparing the ground for new Linux-based applications like Trolltech's Qtopia, the Opera or Access web browsers, etc. The core is free and accepted by the market – applications will follow.

This jibes well with the arguments laid out in Professor Lessig's book. Lessig argues that the early Internet was free, much as Linux is free today. Conceived in open code, the Internet left control to the edge of the network (my Linksys home networking equipment, for example, and the IP software protocols that dictate how my equipment reaches the wider Internet). Everything in the "middle" was left free of control. In similar manner, Linux works because the

[72] This will soon change, however, if Michael Robertson has his way. Apparently itching for another legal battle (after losing $100 million to the recording industry with his last venture, MP3.com), Robertson is taking on Microsoft with his "Lindows" product, a Linux OS that will run Microsoft applications. (*See* http://www.lindows.com for more information.) Just as in the *Sony v. Bleem* case, I would expect that a court will find Robertson's product to be an emulator of sorts, and market-expanding for Microsoft. Hence, I believe (and I hope, anyway) that the courts will be kinder to this Robertson venture than they were to his last.

core (the kernel) is open, with development spurred on by a love of good code.[73] As the platform matures, more and more application developers are encouraged to write code for it, which in turn draws in new participants on the OS/kernel work, which in turn…my point is made.

As noted earlier in this paper, I have had first-hand contact with nearly every major electronics company, and have watched them embrace Linux for their next-generation devices. Sony, Netgear, Mitsubishi, and others are actively looking at moving their entire product portfolios over to Linux. Linux now powers Japan's most successful PDA (the Sharp Zaurus), and has been selling out as quickly as it can ship. Linux will drive Sony's PlayStation, every major home networking box (Linksys, Netgear, D-Link), and every digital set-top box (Motorola's DCT-5xxx, which owns 62 percent of the US market, will be Linux-based, as will Pace Micro's next-generation set-top box, with Pace being the largest set-top box manufacturer worldwide), among other things.

These are but a few examples of market acceptance, but the important thing is that as Linux finds its way onto more and more devices, more and more applications will be written to it. With a developer pool that dwarfs Microsoft's by many times, it seems a conservative bet that robust, market-ready Linux applications will outnumber (as they continue to out-engineer) those of proprietary operating systems by five-to-one within five years. *That* is true innovation.

[73] Eric Raymond makes this argument best, holding that developers code for reasons of ego satisfaction and reputation capital. *See* RAYMOND, *supra* note 52, at 64. I did not accept this argument before, but since my last attempt at addressing Linux and innovation, I have managed a large group of developers at my current employer, Lineo. In my experience, most Linux hackers think most of themselves as coders when they are doing difficult kernel work. Erik Andersen is a classic example, working hours into the night to improve his BusyBox project. (*See* http://www.busybox.net/.)

Inefficiency of the GPL, Revised

I have spent the last few pages repenting in sackcloth and ashes for my former views on the GPL and its effects on innovation, generally, and its effects on application development, specifically. Unfortunately, more penitence remains. I had argued that in addition to its stunting effect on innovation, the GPL is inefficient. The belief was that because the GPL severely restricts the manner in which proprietary code can interact with it, the GPL forces costly designing around it, if one is to attempt to replicate the best in Linux for another OS.

Unfortunately (for my ego), this problem has largely disappeared. As soon as the closed-source world came to understand the GPL, and that it was not locked out of linking to Linux, the design-around problem dissipated. Fortunately, instead of just free-riding on Linux as a public good, closed-source companies have increasingly opened up. Several examples have already been cited; numerous additional examples (with IBM offering many on its own) could be given. It has been amazing to watch.

The GPL: Not Broken, Why Fix It?

The GPL has managed, largely despite itself and Richard Stallman, its creator, to strike a balance between open and closed code to bring Linux to the masses. The market has essentially sidestepped the most obvious interpretation of the GPL – that it is gleefully contaminatory – and decided to believe in Moglen's, rather than Stallman's, interpretation. However, the FSF should codify this reading of the GPL to clarify that, in fact, the GPL allows and encourages producers of code to collect a return on their investment, thereby widening the developer community beyond the free-source developers. In so doing, corporations and individuals will have a clear runway to develop applications, drivers, etc. for Linux, assuring it a place as the world's

dominant operating system. A suggested revision that accomplishes just this is attached as Appendix A. Other alternatives to this approach are discussed below.

BSD-Style GPL?

Even if the GPL is eventually clarified, it is possible that the clarification will not satisfy all closed-source developers, because Stallman is unlikely to go so far as to embrace BSD (Berkeley Software Distribution)-style licensing. Such licensing is the ideal licensing model for some parties, closed- and open-source alike, because it allegedly maximizes freedom, as well as incentives. The BSD allows open-source or closed-source use of its code, and simply requires the user to give credit back to the developer of the original code. This fosters the reputation needs of the developers, and allows for maximum freedom for the derivative work creator. Apache and many other programs (curl, bind, etc.) run under BSD-style licenses. Apache, for its part, delivers approximately 60 percent of all Web pages, according to recent Netcraft surveys,[74] ample evidence that its licensing scheme has not hindered it.

BSD-style licensing overcomes the supposed inefficiencies of both free-source and closed-source development. The potential problems with free source have already been addressed in this paper. On the closed-source side, Shawn W. Potter notes that it breeds inefficiency because it forces developers to 'reinvent the wheel,' as it were, by developing code that others have already created.[75] The BSD sits in the middle, allowing closed-source developers to benefit from open-source software, while still providing the reputation incentive to open-source developers. To close the loop, and bring closed-source software back into the

[74] Microsoft, for its part, only delivers 19.6 percent with its NT servers. *See* Jim Jagielski, *Open Source: Breaking Through the Hype*, WEBTECHNIQUES, Jan. 2001, at 40.
[75] Potter, *supra* note 11, at §49.

commons from which it borrows, a short-term copyright or patent provision could be built into the license. (See below.) Landes and Posner speculate that "there is a level of copyright protection that balances these two competing interests [i.e., Authors' interests to be able to borrow from previous work, as well as to protect their own works] optimally.[76] BSD-style licensing, coupled with tighter time restraints on copyright, seems to offer this ideal.

A former proponent of this model, I am increasingly disenchanted with it. I do not think it goes far enough to ensure a fertile breeding ground in the core of the "Linux network," i.e., the kernel. It allows too much to be consumed by closed-source development. While I see a BSD-style license as a great advancement at the application layer of development, it is less appetizing for the Linux kernel.

Short-Term Copyright

Another alternative would be to introduce short-term copyright provisions into the GPL. Such a system might allow an 18-month (or some other limited period of time) window of closed-source licensing for GPL-derivatives, with the requirement that such code be completely open sourced after the initial closed-source period. There is, of course, no optimal copyright length or breadth (or, at least, no one has developed a scheme with a wide following as yet),[77] but 18 months seems like a reasonable amount of time for a closed-source entity to seed the market with their product and gain traction.[78] My own experience working for an embedded

[76] Landes and Posner, *supra* note 44, at 333.
[77] *See, e.g.,* Howard F. Chang, *Patent Scope, Antitrust Policy, and Cumulative Innovation*, 26 RAND J. OF ECON., Spring 1995; Julie E. Cohen and Mark A. Lemley, *Patent Scope and Innovation in the Software Industry*, 89 CAL. L.R. 1; Vincenzo Denicolo, *Patent Races and Optimal Patent Breadth and Length*, XLIV THE J. OF INDUSTRIAL ECON., Sept. 1996; and Paul Klemperer, *How Broad Should the Scope of Patent Protection Be?*, 21 RAND J. OF ECON., Spring 1990.
[78] Potter notes that Microsoft's OS products historically have a shelflife of two years before a new iteration comes on the market. *See* Potter, *supra* note 11, at §59. While it is certainly true that these "iterations" incorporated code

Linux company suggests that such a period of time would be sufficient to recoup investment of time and resources, or at least to establish a basis for gaining the return on investment.[79] At any rate, I would retain the GPL's current prohibition on closed-source linkage with GPL code, and would stipulate that code which dynamically links with GPL code be open-sourced after a limited period of time, as addressed above. This ensures that all code which benefits from linking to Linux would be made open within a reasonable period of time, enabling more rapid cumulative development.

The devil may well be in the details on this suggestion, but the core idea seems sound. Such a scheme could keep well within the spirit of the GPL, with the focus being to preserve the freedom of all users. Permitting only non-discriminatory licensing and reasonable royalties would accomplish this. Of course, the question will be raised as to how one defines "reasonable." One approach would be to split the world of licensors into two groups: commercial and individual. Commercial entities could be charged whatever the market would bear (i.e., the market would figure out the bounds of "reasonableness"), and the software would be free to non-commercial individuals or entities. Such methodology has a precedent in Troll Tech's licensing of Qt.[80]

In so doing, software innovation would likely increase. In order for companies to continually realize rents from their code, they would need to dramatically speed up the process of innovation, producing new products every 18 months or so. In this way, consumers would be

---

from previous versions of MS-DOS and Windows, complicating the ability to simply open-source them, this is not an insurmountable problem.

[79] Aladdin Enterprises currently deploys a similar model. *See* ERIC S. RAYMOND, *The Magic Cauldron*, *in* THE CATHEDRAL AND THE BAZAAR: MUSINGS ON LINUX AND OPEN SOURCE BY AN ACCIDENTAL REVOLUTIONARY 137, 168-169 (1999).

[80] It must be noted, however, that the open-source world strongly disapproved of Troll Tech's model, eventually pressuring Troll Tech to stop charging for commercial use. One account of the Troll Tech story can be found in Bruce Perens, *The Open Source Definition*, *in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION 171, 175 (1999).

greatly benefited with new code on a regular basis. Software companies would also benefit, because such a development model would help push them toward subscription-based revenue models that would ensure consistent streams of revenue.

Can Open Source Survive Closed-Source Involvement?

The most obvious answer to this question is a resounding "Of course." Open-source software (Linux, Debian, BSD, etc.) has thrived alongside closed-source software, often intermingling with it. Linux aside, Apache, curl, bind, etc. are all licensed under less restrictive Open Source licenses than the GPL, and have continued to thrive despite a more intimate relationship with closed-source code. The evidence strongly favors a belief in software's ability to flourish at the crossroads of open and closed-source licensing.

Free Source advocates like Stallman, however, worry that opening up the GPL to proprietary code will produce a tragedy of the commons. In Garrett Hardin's classic article, "The Tragedy of the Commons," partakers in a public good destroy the good in their efforts to personally benefit from the good. No one has the authority to block others' use of the good, and everyone has the incentive to consume as much of the public good as possible, leading to overconsumption of a finite resource.[81] Stallman might argue that by opening up to closed source, the GPL would invite corporations and individuals to trample on his carefully preserved "commons," always taking, without ever giving back to the community.

However, closed-source involvement in GPL code does not present a tragedy of the commons.[82] Theoretically or in actuality. For one thing, software is not a finite resource. Instead, it is infinitely replicable. In this way, software is arguably a nonrivalrous good. Eric

[81] Garrett Hardin, *The Tragedy of the Commons*, 162 SCIENCE 1243, 1248 (Dec. 1968).
[82] For an alternative view of open source and the tragedy of the commons, see RAYMOND, *supra* note 62, at 149-155.

Raymond talks of everyone grazing freely on the open-source range, with the grass growing greener and longer in consequence. Eben Moglen attempts to contradict this, arguing that taking open code private subtracts from the commons. But how? How does my use of open-source code in any way subtract from another's ability to use it? Moglen is not consistent on this point, given his prior claim that software is different than most intellectual property, because one's use in no way diminishes someone else's.[83]

Open-source software fails to present a tragedy of the commons for another, unrelated reason. As already noted, and as Eric Raymond effectively argues, the currency of open source is not monetary but rather consists of reputation. Hackers code for the love of hacking, and not for pecuniary gain. If this is true, and much ink has been spilled in defense of this theory, then the free-source world is not diminished by a corporation using free-source code for proprietary purposes through dynamic linking mechanisms.[84] If anything, reputation is therefore enhanced, because the developer can point to large corporations as their "clients." The only "tragedy of the commons," then, is that so few people were able to tap into GPL code at first because of the fear it instilled in corporations and individuals who would have otherwise used it.

One counter to this, as James Boyle persuasively argues, lies in the fact the information frontier is *not* quite infinite:

> The information frontier is no more infinite than the West was infinite, and the monopoly property rights now being given out to software and biotech companies rival anything given to the railroad or banking 'trusts' a hundred years ago.[85]

---

[83] For a similar argument, see John Perry Barlow, *The Economy of Ideas: A Framework for Rethinking Patents and Copyrights in the Digital Age (Everything One Know About Intellectual Property Is Wrong)*, WIRED (March 1994) 84, 85.

[84] Raymond also notes that hackers "do not intrinsically object to others profiting from their gifts [software contributions to the open-source community], [but] most…demand that no party (with the possible exception of the originator of a piece of code) be in a *privileged* position to extract profits." In other words, the hacker should also be able to sell her software, not just the closed-source company. *See* RAYMOND, *supra* note 52, 160.

[85] JAMES BOYLE, SHAMANS, SOFTWARE, & SPLEENS: LAW AND THE CONSTRUCTION OF THE INFORMATION SOCIETY 9 (1996).

Boyle argues that granting property rights in information deprives authors of the ability to use such "raw materials" for creating new works.[86]

This is a valid argument, but it is not one that really addresses the Linux phenomenon as it has actually happened. Though the GPL ostensibly says "no" to dynamic linking of closed-source code with GPL code, the market came to interpret this "no" as "maybe," and then (when no response came from the FSF) as "yes." We have had a year to observe the effects of this interaction between closed and open source, and it has indubitably been good. For both sides (closed and open). In fact, it has tended to make converts of the closed-source corporations, and the software world has become more open as a result.

And so we can dispel two further mythical counters to the noninjurious nature of closing off open-source licensing. One is that such use discourages others to contribute their code. Unfortunately, history has not been kind to this belief. Apache provides the classic counterexample. Despite Apache's express decision to allow closed-source development of its code base, and despite its open arms to IBM and other corporations who wanted to exploit the Apache code, open-source developers have continued to contribute even when they know companies like IBM will not necessarily open source their contributions.[87] Regardless, such reasoning mistakes the probable incentives hackers have for contributing – most developers arguably do so to build reputation capital, as Raymond argues, and which has been noted above.

Another counterargument is that closed-source users will innovate from the open-source base, but will not release these innovations to the wider public, thereby yielding no benefits to

---

[86] *Id*., at 57.

[87] Presentation given at Stanford Law School by Brian Behlendorf, cofounder of the Apache Web Server Project (April 11, 2001). Behlendorf argues that while it is important to keep the network infrastructure of the software world open, because this is where commonality/interoperability is crucial, everything on top of this can be closed. Proprietary software, then, has its place higher up the value chain, i.e., closer to the consumer. *See also* LARRY LESSIG, CODE AND OTHER LAWS OF CYBERSPACE 207 (1999) (arguing that software is increasingly written within companies, with potentially deleterious effects on privacy and freedom).

the wider community. In the first place, this argument misses the likely possibility that such individuals would never contribute their code in the first place if they must GPL it. So, such code would not find its way into the commons, anyway. But more importantly, this argument has not held up in practice. Examples were cited above as to the opening of the software industry, as it embraces Linux. The free rider problem has frankly not been much of a problem.

Conclusion

Linux has won, both because of and in spite of the GPL. The market statistics increasingly evidence this, and the trend shows no signs of abating. Though the GPL barked a lot, developers learned over time that it did not often bite. In response, closed-source developers initially experimented with linking their proprietary code to GPL code through dynamic linking. With that experiment successful, they have increasingly opted to open-source core and non-core technologies. In addition, a trend has emerged with software as a service, and not as a product, which has made the transition to GPL Linux software all the easier.

This transition has Microsoft scared, and should have consumers cheering. Software is finally nearing the era when consumers will write for consumers, rather than the standard era of businesses writing code for businesses. With an open development 'network' in place, no one entity can derail innovation in a narrow, selfish direction. Software, thanks to the GPL, is on the cusp of truly belonging to us all: a true public *good*.

Appendix A: Suggested Revision of the GNU GENERAL PUBLIC LICENSE

See separate attachment