ADVENTURES IN SOFTWARE LICENSING: SCO V. IBM AND THE FUTURE OF THE OPEN SOURCE MODEL

ANDREW LAFONTAINE [*]

[*] Andrew Lafontaine is a J.D. candidate at the University of Colorado (2006).

INTRODUCTION

People write for many reasons. Some for pleasure, others for money. An author wishing to profit from their work must find some way to limit access to that work to customers willing to pay for the privilege.[1] To accomplish this, authors typically employ a two-pronged strategy of copyright and license. The copyright prong imposes a statutory prohibition on the copying, use, or modification of the work without the copyright holder's permission.[2] The license prong is a contractual grant of that permission, subject to a set of restrictive terms designed to prevent the licensee from diluting the copyright holder's monopoly.[3] By lifting copyright prohibitions in exchange for money, this arrangement (which I label the "proprietary model") allows the author to earn a profit from their creation, which in turn incentivizes them to produce new works. The proprietary model of copyright and restrictive license is particularly prevalent in the software industry: companies like Microsoft license their programs to end users for a fee, using the proceeds to pay for the development of the next version of "Windows" or "Office."

Recently, however, another model software development known as the "free/open-source software" (F/OSS) movement has emerged to challenge the dominance of the proprietary model. This new model turns traditional notions of limited access on its head by inviting interested programmers from all over the world to freely copy, share, and modify each other's work.[4] Thanks in no small part to the advent of the Internet,[5] numerous projects have sprung up to develop F/OSS

1. PAUL GOLDSTEIN, GOLDSTEIN ON COPYRIGHT § 1.14.1 (3d ed. 2005).
2. 17 U.S.C. § 106 (2004).
3. The restatement describes a license by stating: "In a broad sense, the word 'license' is used to describe any permitted unusual freedom of action. It may be used to describe privileges to carry on businesses or to practice callings not otherwise permitted." RESTATEMENT OF PROP. § 512 cmt. a (1944).
4. Software development based on the free sharing of code is not new—during the early days of the computer era, code sharing was a regular practice in programming hotspots like AT&T's Bell Laboratories and in academia. However, it became increasingly rare as these institutions started to recognize the potentially enormous value of the software they produced.
5. Writing about the Linux Project, Columbia Law Professor Eben Moglen noted that: "[T]he Internet made it possible to aggregate collections of programmers far larger than any commercial manufacturer could afford . . . in a development project ultimately involving more

applications, often with the express purpose of competing with their commercially developed counterparts. Anyone may volunteer to participate in these projects in whatever capacity they desire; those who make contributions to such projects are rarely paid for their services, and the resulting products may be freely used by anyone. This approach has several advantages, not the least of which is that the presence of many sets of eyes ensures that software "bugs" (errors or glitches in the coding process) will be quickly discovered and corrected.[6]

F/OSS development has attracted a great deal of support and media buzz in recent years, yet despite all this attention the movement remains difficult to define precisely. F/OSS advocates are not of one mind—there is a great deal of disagreement about things as basic as the meaning of terms like "free" and "open." However, one thing that is widely agreed on is that the best way to promote the unfettered use and widespread distribution of code is to avoid restrictions on access to source code wherever possible — a position anathematic to the proprietary model. As important as this is to the open source model, F/OSS development is not synonymous to placing the code in the public domain. As I will explain below, F/OSS is typically distributed under a license which has been specially crafted to ensure that the code remains accessible long after it has left the developer's hands.

One especially important piece of F/OSS, which will be the focus of this paper, is the Linux operating system (OS).[7] Originally begun as a research project, Linux is widely seen as a successor to the venerable Unix OS, a proprietary system on which it is based. Although highly regarded for its power and stability, Unix fractured into a number of incompatible variants in the 1970s and 1980s, making development difficult.[8] By providing a single platform on which to standardize, while at the same time maintaining the familiar conceptual structure of Unix, the Linux project has been a great success.

than one million lines of computer code — a scale of collaboration among geographically dispersed unpaid volunteers previously unimaginable in human history." Eben Moglen, *Anarchism Triumphant: Free Software and the Death of Copyright*, FIRST MONDAY (1999), http://www.firstmonday.org/issues/issue4_8/moglen/index.html.

6. In his seminal paper, The Cathedral and the Bazaar, open source advocate Eric Raymond coined a phrase which has become the open source movement's unofficial mantra—"Given enough eyeballs, all bugs are shallow." Eric Raymond, *The Cathedral and the Bazaar, in* THE CATHEDRAL AND THE BAZAAR (Feb. 06, 2006) (unpublished manuscript, available at http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/).

7. Linux was and continues to be developed under the Free Software Foundation's "General Public License" (GPL), endowing it with a special feature known as "copyleft." While Linux is by no means unique in this respect, it should be pointed out that copyleft embodies the tenets of a particularly strident branch of open source philosophy.

8. *See* James V. DeLong, *The Enigma of Open Source Software (Version 1.0)*, 11.8 PROGRESS & FREEDOM FOUND. 12 (2004), http://www.pff.org/issues-pubs/pops/pop11.8opensource.pdf.

Recently however, a controversy has arisen between IBM, one of the major backers of Linux, and a small software company named SCO. As holder of the copyrights for one of the many Unix variants, SCO has sued IBM for allegedly inserting copyrighted code into Linux, an action strictly prohibited by IBM's Unix license agreement with AT&T (SCO's predecessor in interest.

Linux is widely seen as the greatest achievement of the F/OSS model, so this litigation raises a number of interesting questions. What is the likelihood of a SCO victory? What are the consequences for the Linux project if SCO is victorious? Are other open source projects vulnerable to similar challenges? This paper seeks to provide answers to these questions. My position is that while allegations of copyright violation in theory present a serious threat to open source projects like Linux, SCO is unlikely to prove wrongdoing in this particular litigation. I also argue that just as open source developers should not be allowed to take proprietary code without authorization (as is being alleged in the Linux suit), proprietary developers should not be allowed to appropriate the hard work of their open source counterparts. The open source movement's adoption of strategies like the General Public License (GPL) will help ensure that it is not put at a competitive disadvantage vis-à-vis proprietary software.

This paper is structured as follows. Section I sets out the basic principles of copyright and licensing, explaining the mechanics of the law and the concepts of original and derivative works. Section II describes how these copyright and licensing schemes are put to use by the proprietary and open source models, with particular focus on the specific provisions of the GPL. Section III traces the origin of the Unix and Linux operating systems, each of which represent a practical application of the two software models. Section IV covers the specific aspects of the current litigation between SCO and IBM, starting with a list of SCO's allegations, continuing with a discussion of license terms constraining IBM, and concluding with an analysis of IBM's position. Finally, Section V discusses the implication of this suit for the future of open source development, as well as the importance of the GPL in maintaining a level playing field between the open source and proprietary models.

I. COPYRIGHTS AND LICENSING

A. The Basics

The U.S. Constitution grants Congress the power to "promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries."[9] For authors, Congress has provided the ability to protect unique writings through the mechanism of copyright. Under federal law, no one except the copyright owner may reproduce, prepare derivative works based on, or distribute copies of copyrighted works.[10] These rights attach to "original works of authorship fixed in any tangible medium of expression"[11] and are vested initially in the author of such works.[12]

Due to the prohibitions set forth by copyright law, a non-copyright holder who wishes to reproduce, distribute, or prepare derivatives of a copyrighted work can only do so by obtaining the copyright holder's permission.[13] This is commonly accomplished through the mechanism of a license, which is a contract allowing the licensee to take certain actions that would otherwise be prohibited by copyright law.[14] Once granted, any action specifically provided for by the license will no longer be in violation of copyright; however, actions not provided for are still prohibited.[15] While some licenses grant the licensee virtually unrestricted access to the work,[16] it is more common for a license to selectively lift only certain copyright restrictions and leave others in place. The license may, for example, allow the creation of verbatim copies but not derivative works, or the preparation of a derivative work but not its distribution.

As a contract, a license can take just about any form the parties desire, so the grant of rights is often conditioned on the licensee's acceptance of other terms. While these terms commonly specify how the licensee may use the licensed item, they are by no means limited to this. A license might require the licensee to pay a fee for each copy of the work they make, or promise not to display the work in certain ways, or even (to give a somewhat ridiculous but theoretically possible example) shave her head before she is granted the right to use the work under license.[17]

9. U.S. CONST. art. I, § 8, cl. 8.

10. *See* 17 U.S.C. § 106 (2004).

11. *Id.* at § 102.

12. *See id.* at § 201.

13. *See id.* at § 106.

14. "A license is, in legal contemplation, merely an agreement not to sue the licensee for infringement." DAVID NIMMER & MELVILLE B. NIMMER, NIMMER ON COPYRIGHT § 10.01 n. 73.1 (Lexis Ed. 2004) (hereinafter NIMMER).

15. "[I]n the absence of clear evidence of a contrary intent, where an assignee has prepared the assignment, rights not expressly granted will often be held to be reserved by the assignor." NIMMER, *supra* note 14, at §10.08. Even though Nimmer refers to an assignment of copyright rather than a license, the principle is basic to contract law and applies in both contexts.

16. For instance, the BSD license allows the licensee to use, modify, and/or redistribute the code, subject only to a warranty disclaimer and a requirement that copyright notices be retained. *See infra* note 81.

17. A real world example of licensing terms so harsh they border on the absurd are those

A valid license only protects the licensee from copyright infringement if two conditions are met. First, the licensee must not act outside the scope of her license. As explained above, any action not specified in the license is still barred by copyright law. Second, she must observe any and all license provisions.[18] If the license is granted on certain terms, failure to comply with those terms may automatically revoke the license.[19] For this reason, license terms are vitally important, since revocation puts the licensee in the same position she would have been in had she never obtained the license in the first place—she is no longer shielded from the prohibitions of copyright law and any further use, distribution, or modification of the licensed work is illegal. It also means that copyright violation often goes hand-in-hand with breach of contract.[20]

B. Computer Software

Software is available in two forms: source code and object code. Source code is the form in which software is originally written — it is human readable such that a programmer can understand it and modify it if she wishes.[21] Object code is a machine readable form that the source code must first be translated (or "compiled") into before it will run on a computer.[22] A programmer who hopes to make useful modifications to a piece of software must have access to its source code, and it is very difficult (if not impossible) to re-translate object code back into source code.[23] Proprietary software publishers, who rely on limited access for their business model, consider their programs' source code to be the crown jewels of their intellectual property and guard it jealously. They typically only distribute their programs in object code format, and forbid

once used by a company called Man's Best Friend Software. Its products, marketed to dog breeders and kennel owners, required the licensee to not to publish "derogatory statements" about the company, its products, or its employees. In addition, the terms required that the licensee "agree" that such statements would cause the company to suffer inestimable harm. In lieu of a trial or hearing on damages, they were obligated to pay a fixed amount of $10,000 for each derogatory publication. *See* Edward Foster, *The Worst Sneakwrap Agreement*, *at* http://www.gripe2ed.com/scoop/story/2004/3/4/84017/93009 (last visited Mar. 19, 2005).

18. *See* NIMMER, *supra* note 14, at § 10.15.

19. If the violation consists of a failure to satisfy a condition precedent to the grant of the license, the work was never effectively licensed in the first place and hence copyright violation is automatic. Alternatively, when dealing with license covenants that are not conditions precedent to the grant of a license, the license may provide for automatic revocation upon a breach by the licensee. *See id.*

20. *See id.*

21. *See* Wikipedia, Source Code, http://en.wikipedia.org/wiki/Source_code (last visited Mar. 26, 2005).

22. *See id.*

23. *See* Wikipedia, Decompiler, http://en.wikipedia.org/wiki/Decompiler (last visited Mar. 26, 2005).

users from attempting to de-compile or reverse engineer their program. The open source model, on the other hand, gets its namesake from the fact that this code is available to all.

Computer software is rarely bought and sold outright. Someone who buys a piece of boxed software from a store (or over the Internet) has in all likelihood purchased a license granting them the right to make a copy of the program to use on their own computer, on the condition that they agree with a specified set of terms. The typical proprietary license prohibits the licensee from making unauthorized copies of the software for distribution, while the typical open source license expressly allows the licensee to distribute copies freely.[24]

1. The Special Problem of Derivative Works

Software is frequently licensed not only for its usefulness as a standalone product, but also for the purpose of creating derivative works. A derivative work is nothing more than a recasting, transformation, or adaptation of one or more preexisting works.[25] The popular treatise *Nimmer on Copyright* states: "Copyright in a derivative or collective work covers only those elements contained therein that are original with the copyright claimant."[26] This means that two sets of rights exist in a derivative work. Copyright for the underlying elements of the original work remain with the original author, while copyright for the new contributions belong to the derivative author.[27] For this reason, the original author may not claim copyright in the new contributions, since she did not write them, but she may still claim copyright in the elements of the underlying work that the derivative contains.

Computer programs are a fertile ground for the creation of derivative works. Rather than writing a new program from scratch, it is often much easier to take existing code and modify it to provide new functionality.[28] One need look no further than popular applications such

24. As noted in section III(B)(ii), *infra*, some open source licenses provide that any redistributions *must* include the source code, while others make this optional.

25. *See* H.R. REP. No. 94-1476, at 57 (1976); *See also* 17 U.S.C § 101 (2004). Note that a derivative work is different from a collective work, which applies to items "such as a periodical issue, anthology, or encyclopedia, in which a number of contributions, constituting separate and independent works in themselves, are assembled into a collective whole." 17 U.S.C. § 101.

26. NIMMER, *supra* note 14, at § 3.04.

27. This of course assumes that the derivative author has obtained permission from the original author to create a derivative work in the first place. An unauthorized derivative author has no right to create a derivative work. *See* 17 U.S.C. § 106 (2004).

28. As Eric Raymond puts it: "Good programmers know what to write. Great ones know what to rewrite (and reuse)." Eric Raymond, *The Mail Must Go Through*, *in* The Cathedral and the Bazaar (unpublished manuscript, available at http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ar01s02.html)

as Adobe Acrobat (currently in its seventh incarnation) or the Microsoft Office suite of programs (eleventh) to see this process in action. In copyright terms, each is a derivative of the previous version.[29]

Derivative works can be created in two ways. The first (and more straightforward) way occurs when someone adds to or modifies an existing work. This scenario occurs all the time in computer programming. A developer might start with a piece of code which is useful for completing certain tasks, but which does not quite fulfill her needs, and add and subtract code until it does.

The second method for creating derivative works is far more complex, and may be unique to the world of software. A completely original piece of code might be considered derivative if it interacts heavily with a pre-existing program, such as where one software module invokes the use of another.[30] This commonly occurs in the context of software "libraries," which, although not free-standing executables themselves, contain subprograms and helper data that other programs can utilize.[31] Depending on how a program invokes a copyrighted library, it might be considered a derivative work for the purposes of copyright law.[32]

2. Sublicensing

The author of a derivative work who wishes to distribute it as a standalone product may not do so unless they have the permission of the

(discussing his own experience improving email clients).

29. Note that in these examples, the party doing the updating also happens to be the copyright holder. However, in theory there is no reason why "Acrobat 8" or "Office 12" could not be released by a third party (assuming of course that they had proper authorization).

30. For an excellent discussion of this confusing but important topic, see David McGowan, Legal Aspects of Free and Open Source Software (Oct. 5, 2004) (unpublished manuscript, *available at* http://www.law.umn.edu/uploads/images/253/McGowanD-OpenSource.rtf; DeLong, *supra* note 8, at A1. Although this understanding of derivative is controversial, that has not stopped the Free Software Foundation from taking such a position. *See* Free Software Foundation, Frequently Asked Questions about the GNU GPL, http://www.gnu.org/licenses/gpl-faq.html (last visited Feb. 06, 2006).

31. See Wikipedia, Library (computer science), http://en.wikipedia.org/wiki/Shared_library (last visited Mar. 21 2006); BookRags.com, Software Libraries Summary, http://www.bookrags.com/sciences/computerscience/software-libraries-wcs.html (last visited Mar. 23, 2006).

32. As the preamble to The Free Software Foundation's "Lesser GPL" (LGPL) points out: "When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library." Free Software Foundation, GNU Lesser General Public License, http://www.gnu.org/licenses/lgpl.html. (last visited Mar. 14, 2006). Software libraries are an extremely important programming tool, and this rigid rule can create complications when dealing with certain "copyleft" provisions found in the standard GPL (set out in greater detail below). In recognition of this fact, the LGPL was created to allow open source developers the option of allowing proprietary programs to link to their libraries without running afoul these provisions. *See id.*

original author. Even though the derivative author holds the copyright for their contributions to the work, copyright for the underlying elements still rests with the original author. Permission is commonly obtained through a special form of license known as a sublicense, whereby the original author (the sublicensor) grants the derivative author (the sublicensee) permission to distribute a work in which she only holds partial copyright. Like any other license, a sublicense can set the terms on which this permission is granted, and it is common for these terms to affect the way the sublicensee distributes the derivative work to third parties. For example, if X sublicenses code from Y for the purpose of creating a derivative work, Y's sublicense agreement may dictate some or all of the terms on which X may license the derivative to Z.

II. OPEN SOURCE VERSUS PROPRIETARY SOFTWARE

A. Two Alternative Models of Software Development

Copyright law is premised on the idea that the creation of written works is driven by the pursuit of profits[33]—profits which the author derives from his ability to act as gatekeeper to his work.[34] Without this control, authors would be unable to make money off their creations and would lose their motivation to produce new ones.[35]

Combined with restrictive licensing terms, copyrights enable authors to sell limited rights to copy their work without giving up their monopoly status. In this proprietary model of development, copyrights are strictly enforced and the right to copy, use, or distribute code is granted only for a price. The more rights granted, the higher the price. Moreover, authors keep a tight grip on their work by crafting detailed licensing agreements to ensure that the rights granted under it extend no further than the immediate licensee. Sublicense agreements enabling the distribution of derivative works may require additional payment.

Over the past decade or so, a growing number of computer programmers, technology enthusiasts, and scholars have begun to

33. To quote Samuel Johnson, "No man but a blockhead ever wrote, except for money." JAMES BOSWELL, THE LIFE OF SAMUEL JOHNSON, L.L.D 19 (G. B. Hill ed., Oxford: Clarendon Press 1934).

34. Note that a copyright holder's ability to control access is not perfect. Copyright does not prohibit someone from reading a copyrighted book (or even selling it to someone else after they have done so)—only from copying or distributing it. However, in the context of computer software, the ability to copy is synonymous with access as a practical matter. Software is useless until it has been copied from the medium on which it has been distributed (such as a CDROM) to the machine were it will be run. The same goes for code available over a network such as the internet.

35. *See generally* Andrew Beckerman-Rodau, *Are Ideas Within The Traditional Definition of Property?: A Jurisprudential Analysis*, 47 ARK. L. REV. 603 (1994).

question whether this approach is the best way to maximize the output and value of creative work.[36] Restricting the ability to copy, modify, and distribute software may allow the copyright holder to extract the maximum amount of profit from their work, but that does not always guarantee the highest quality product.[37] Recently there has been a great deal of interest and participation in collaborative software development projects, characterized by wide distribution of code and freedom for users to copy, modify, and distribute it as they wish.[38] This drive toward collaborative development is often referred to as the open source movement and the term F/OSS refers to programs developed under the open source banner. As noted earlier, code sharing has been around since the dawn of the computer era, but this new movement stands apart both in its purposeful approach and its global reach.

F/OSS development projects are best conceived of as being structured as concentric rings: those with greater interest and willingness to commit more time occupy the center, while dabblers remain at the fringes.[39] This amorphous form is in keeping with the general egalitarian and equalitarian ideals of the movement, although some have suggested these structures are much more hierarchical in practice.[40] F/OSS projects have traditionally been supported by volunteer efforts, but an untold amount of money and programmer time has been contributed by commercial enterprises as well — IBM alone has committed over 200 programmers and invested over one billion dollars to the Linux project.[41]

36. *See* Yochai Benkler, *Coase's Penguin, or, Linux and the Nature of the Firm*, 112 YALE L.J. 369 (2002).

37. Few issues arouse as much passion as the dispute over whether the F/OSS model or the proprietary model produces "better" software. Part of the problem is there is no single yardstick to measure this by. F/OSS programs are notoriously user-unfriendly, but at the same time, they are more customizable to the user who knows what they are doing. Gauging by the number of software bugs is similarly problematic. In their 2005 year-end index, Cyber The US Computer Emergency Readiness Team identified 812 Windows operating system vulnerabilities, compared with 2328 for Unix/Linux operating vulnerabilities. *See* Cyber Security Bulletin 2005 (Dec. 29, 2005), *available at* http://www.us-cert.gov/cas/bulletins/SB2005.html. However, a 2004 report published by the online journal The Register noted a marked imbalance in the relative severity of the vulnerabilities and concluded that Linux was more secure. *See* Nicholos Petreley, *Security Report: Windows vs. Linux*, THE REGISTER, Oct. 22, 2004, http://www.theregister.co.uk/security/security_report_windows_vs_linux/. I do not take a position in this debate, and merely note that F/OSS advocates are united in their belief that their model is methodologically oriented to producing the highest quality code (even if it does not always deliver on that promise) and that the proprietary model necessarily subordinates this goal to the producer's bottom line.

38. *See* David McGowan, *Legal Implications of Open-Source Software*, 2001 U. ILL. L. REV. 241, 241-42 (2001).

39. *See* DeLong *supra* note 8, at 36.

40. *See id.* at 34.

41. IBM CORPORATION, THE LINUX SERVICES OPPORTUNITY 16-17 (2003), http://www.ibm.com/partnerworld/pwhome.nsf/vAssetsLookup/ci_LinuxServices02.pdf/$File

There are probably as many different beliefs about the meaning of the term "open source" as there are programmers who espouse it. Some feel it is an embodiment of the American commitment to free expression, while others see it as a way to oppose monolithic software companies (Microsoft in particular) that currently dominate the world of computer software publishing.[42] For more than a few, it has taken on philosophical overtones, implicating issues that have as much to do with natural rights as they do with economic theory.[43] The very term "F/OSS" reflects this multiplicity of views: its most fervent adherents preach the gospel of "free software,"[44] while those advocating "open source software" are more pragmatically oriented.[45]

Despite all this, there are general characteristics of F/OSS that set it apart from proprietary software. In particular, the open source development model has different goals in mind from the proprietary code model. For all its ideological underpinnings, the primary objective of F/OSS development is quite functional: to produce the best code possible by allowing many individuals to participate in its creation. Software of all kinds is often plagued by coding errors called "bugs," which cause programs to malfunction or crash. A key component of the open source philosophy is that the best way to produce well written and bug-free software code is to allow a large number of programmers to tinker with it.[46]

In a typical open source project, a programmer, or a small group of programmers working together, posts unfinished program code to the Internet, where it can be downloaded by anyone. Other programmers and computer hobbyists then donate their time, completing unfinished sections and poring over the code trying to identify potential flaws. Working portions of code are run under adverse conditions to see if they fail, and if so, how. Not only does this process help uncover problems

---

/ci_LinuxServices02.pdf.

42. *See* Richard Stallman, *Is Microsoft the Great Satan?, in* PHILOSOPHY OF THE GNU PROJECT (2000), http://www.gnu.org/philosophy/microsoft.html.

43. *See* Richard Stallman, Why Software Should Not Have Owners, in PHILOSOPHY OF THE GNU PROJECT (2005), http://www.gnu.org/philosophy/why-free.html; Richard Stallman, *Why Software Should Be Free, in* PHILOSOPHY OF THE GNU PROJECT (1992), http://www.gnu.org/philosophy/shouldbefree.html.

44. Richard Stallman, one of the free software movement's most vocal advocates, puts things this way: "Free software' is a matter of liberty, not price. To understand the concept, you should think of "free" as in "free speech," not as in "free beer." Free Software Foundation, *The Free Software Definition, in* PHILOSOPHY OF THE GNU PROJECT (2004), http://www.gnu.org/philosophy/free-sw.html.

45. *See* Eric Raymond, Keeping an Open Mind (2003), http://www.catb.org/~esr/writings/openmind.html. Professor David McGowan provides a particularly clear picture of the dispute between firebrands like Stallman and more moderates voices like Raymond. *See* McGowan, *supra* note 38, at 261-63.

46. *See* Raymond, *The Cathedral and the Bazaar*, *supra* note 6.

that might otherwise have been overlooked, it allows a development project to draw on a much larger pool of expertise to find solutions. This process of peer review, well known in other fields, carries with it an added benefit captured by the old adage "many hands make like work."

It is the task of the open source developer to coordinate these efforts and keep everything running smoothly. While proprietary developers typically work in isolation, open source developers engage in a continuing dialogue with the programming community. Depending on the project, its coordinators may do very little of the actual coding themselves, instead devoting most of their time to managing logistics.[47] Code must be swapped back and forth, new contributions must be evaluated and incorporated, and the updated versions must be redistributed so the process can begin again.

Unlike the proprietary model of software development, the open source model is not explicitly geared toward inducing programmers to contribute with the promise of financial reward. While some programmers (such as those employed by commercial firms engaged in F/OSS development) do get paid for support, the model is premised on a number of non-monetary motivations. In a world where coding is seen as "a gift and an expression of art,"[48] some contribute out of a sense of altruism,[49] while others do it for the sheer enjoyment of solving complex problems and participating in something larger than themselves.[50] Professor Benkler, who has analyzed this issue at some length, breaks the motivations of F/OSS developers down into three categories: monetary rewards (which can be substantial for service-type contracts), intrinsic hedonic rewards (a sort of "play ethic"), and social-psychological rewards (pleasure from increasing one's status in the eyes of others through meaningful contributions to a project).[51] Whether these motivations are sustainable is an open question, but with open source projects like Linux still going strong after 15 years, they do not appear as transitory as some

47. In describing the work of Linus Torvalds (the creator of the open source Linux OS, discussed below) Eric Raymond comments that Mr. Torvalds' greatest contribution was not the Linux program itself, but his masterful use of the open source model. Says Mr. Raymond, "When I expressed this opinion in his presence once, he smiled and quietly repeated something he has often said: 'I'm basically a very lazy person who likes to get credit for things other people actually do.'" *Id.*

48. ERIC RAYMOND, OPEN SOURCE INITIATIVE, OSI POSITION PAPER ON THE SCO-VS.-IBM COMPLAINT (2004), http://opensource.org/sco-vs-ibm.html#id3153667.

49. *See* Richard Stallman, *Self Interest, in* PHILOSOPHY OF THE GNU PROJECT (2002), http://www.gnu.org/philosophy/self-interest.html; Alfie Kohn, *Studies Find Reward Often No Motivator*, BOSTON GLOBE, Jan. 19, 1987, *available at* http://www.gnu.org/philosophy/motivation.html.

50. *See* Eric Raymond, *Homesteading the Noosphere*, *in* THE CATHEDRAL AND THE BAZAAR (Aug. 24, 2000) (unpublished manuscript, available at http://www.catb.org/~esr/writings/cathedral-bazaar/homesteading/).

51. *See* Benkler, *supra* note 36, at 423-35.

would argue.[52]

B. Open Source Licensing

As explained above, F/OSS programs are not typically placed in the public domain, but are instead distributed under special licenses designed to promote (rather than restrict) access to the underlying source code. Of these, the most well known, and probably the most controversial, is the GNU General Public License (GPL), under which Linux is distributed. A great deal of misunderstanding surrounds the GPL, so the next two subsections aim to clear up this confusion. I will first outline the basics of the GPL, then address some of its more exotic features, and finally explain the relationship between the GPL and the open source movement in general.

1. The General Public License and "Copyleft"

The open source model requires unfettered access to source code in order to function. To this end, MIT researcher Richard Stallman created the GPL as a mechanism for software authors to provide such access.[53] Most licenses rely on the threat of copyright enforcement to keep works closed off, but in a form of legal jujitsu,[54] the GPL uses licenses to open works up to the world. Weighing in at around five single spaced pages of text, the GPL is short in comparison to many proprietary licenses.[55]

A quick glance at § 1 of the GPL reveals that it does not contain the types of restrictions contained in a typical proprietary license. It allows users to copy and distribute source code free of charge, so long as they provide to distributees a notice of copyright, a warranty disclaimer, and a full copy of the GPL license terms.[56] To facilitate improvement of programs, § 3 states that anyone who releases a program in object code format must also provide source code, or else make it easily accessible.[57]

If the GPL's purpose was merely to allow licensees to take code on generous terms, it would be rather uninteresting. However, the GPL takes the concept of open access a step further by incorporating a

52. For a more critical view of the longevity of the open source movement in general, *see* DeLong, *supra* note 8.

53. LI-CHENG TAI, THE HISTORY OF THE GNU GENERAL PUBLIC LICENSE (2001) http://www.free-soft.org/gpl_history/.

54. The term "legal jujitsu" comes from Benkler, supra note 36, at 446. This fighting style's art of using an opponent's momentum against them is an apt metaphor for the mechanics employed by the GPL.

55. FREE SOFTWARE FOUNDAITON, GNU GENERAL PUBLIC LICENSE (1991), http://www.gnu.org/licenses/gpl.html.

56. *Id.*

57. *Id.*

mechanism to ensure that open source software, once licensed, remains open. In § 2(b), the GPL provides that licensees may only take code if they agree to re-license any resulting work (provided they choose to distribute it) under the GPL as well.[58] In other words, any time code licensed under the GPL is distributed, either as a standalone product or as a component of a derivative work, the resulting product being distributed must itself be licensed under the GPL. Failure to do so will void the license, as provided in § 4.[59] In this way, the GPL perpetuates itself from program to program, creating a copy in each new iteration of the code originally licensed. Lest the licensee be caught unaware, this critical feature of the GPL is spelled out explicitly in § 6.[60]

A licensee who uses or modifies a GPL licensed program need not distribute it. If the licensee merely wants the program for their own personal use, they are not required to re-license it. The self-perpetuating features of the GPL are only triggered by the distribution of the program (or any modifications thereto).[61] Should a potential licensee be unwilling to re-license the program on the GPL's terms, § 5 instructs them not to license the program in the first place.[62]

To better understand how the GPL works, consider the following illustration. X creates a small piece of software and makes it available under the GPL. Y wants to incorporate this code into another program that she is working on, so she takes the code under the terms of the GPL, which authorizes derivative works. If Y chooses to distribute her new program containing X's code, Y's program must itself be released under § 2(b) of the GPL. Conversely, Y is barred from using X's code if Y does not want to distribute it under the GPL (perhaps she hoped to sell it for a profit).

The self perpetuating nature of the GPL has led some to label the license as "viral."[63] This viral nature is by design: it prevents programmers from incorporating a piece of code originally developed under an open model into a proprietary work. Many feel that allowing software developers to take from the programming community without giving back anything defeats the purpose of the open source model.[64] A

58. *Id.*

59. *Id.*

60. *Id.*

61. Because it allows the redistribution of the licensed work or a derivative of it, the GPL can be thought of as both a license and a sublicense rolled into one.

62. GNU GENERAL PUBLIC LICENSE, *supra* note 55.

63. Microsoft, which views open source software as a threat, is particularly fond of this somewhat loaded characterization. *See* Butt Wai Choon, *Not Quite an Open-and-Shut Case* (Mar. 2002), http://www.microsoft.com/malaysia/business/articles/linkpage3866.asp.

64. *See* Andrea Ciffolilli, *The Economics of Open Source Hijacking and the Declining Quality of Digital Information Resources: A Case For Copyleft*, 9 FIRST MONDAY (Sept. 2004), http://www.firstmonday.org/issues/issue9_9/ciffolilli/index.html.

non-viral GPL would be powerless to prevent the licensee from re-releasing the code under whatever onerous terms they wished. Since Stallman views freedom as an important component of open source software, a viral license is necessary to preserve that freedom for others: "Someone who uses your code in a non-free program is trying to deny freedom to others, and if you let him do it, you're failing to defend their freedom."[65]

Stallman coined the term "copyleft" to describe this protection of open source software through viral licensing. "Proprietary software developers use copyright to take away the users' freedom; we use copyright to guarantee their freedom. That's why we reverse the name, changing 'copyright' into 'copyleft'."[66] Software that has been copylefted is not only available to the public for free—its availability is forever protected through the unorthodox use of copyright law. This feature is the key innovation of the GPL.[67]

2. Does Open Source Necessarily Imply Copyleft?

While the GPL plays a vital role in the open source movement, a particular program need not be licensed under the GPL to be considered open source. The model includes any code developed under principles of free access and modification.[68] The particular licensing regime a developer chooses is merely a mechanism to put those principles into practice. There are literally dozens of standard licenses that an open source developer may choose from,[69] and if none meet the developer's needs, she is always free to write her own.

In an effort to explain just how a license should be structured to ensure the implementation of open source principles, a non-profit group called the Open Source Initiative (OSI) has promulgated a set of ten characteristics required of a license before software provided under it can be considered truly open source.[70] Copyleft figures prominently in this list. Though the status of copyleft as the *sine qua non* of the model is open to debate, it is widely regarded as a central feature of open source

65. Richard Stallman, *Why Copyleft, in* PHILOSOPHY OF THE GNU PROJECT (2003), http://www.gnu.org/philosophy/why-copyleft.html.

66. FREE SOFTWARE FOUNDATION, LICENSES (2005), http://www.gnu.org/licenses/licenses.html#WhatIsCopylefte.

67. At least one scholar has raised questions about the legal validity of copyleft provisions on privity of contract grounds. While the GPL has never been tested in court, McGowan argues that this is not a serious threat. *See supra* note 38, at 289-303.

68. Richard Stallman, *The Free Software Definition, in* PHILOSOPHY OF THE GNU PROJECT (2006), *http://www.gnu.org/philosophy/free-sw.html* (last visited Mar. 14, 2006).

69. *See* THE FREE SOFTWARE FOUNDATION, VARIOUS LICENSES AND COMMENTS ABOUT THEM, http://www.gnu.org/licenses/license-list.html (last visited Feb. 7, 2006).

70. THE OPEN SOURCE INITIATIVE, THE OPEN SOURCE DEFINITION (2006), http://www.opensource.org/docs/definition.php.

development.[71] The inclusion of copyleft terms in OSI's open source definition, combined with the fact that copyleft was pioneered by the GPL, helps explain the common misconception that code can only be considered open source if it is licensed under the GPL.

III. PUTTING THE DIFFERENT DEVELOPMENT MODELS INTO PRACTICE: UNIX AND LINUX

Explaining the current controversy between SCO and IBM requires at least a basic understanding of the Unix development's extremely complex history. Unix does not have a unitary identity: [72] over the years, hundreds of variants have been developed, often by companies and institutions with very different agendas.[73] Portions of these variants have been mixed and mingled, and the result is a family tree that has aptly been described as a "plate of spaghetti."[74] This section only covers the few variants involved with Linux and the SCO v. IBM suit, but it is important to note that the disputes engendered by the incompatibility of the various Unix versions are a large part of Linux's *raison d'être*, and the agreements that came out of this period will have a great impact on the outcome of the present litigation.

A. General Background

An operating system (OS) is a very important piece of computer software that controls the hardware of a specific data-processing system in order to allow users and application programs to make use of the system.[75] There are many different operating systems available today, of which the Microsoft Windows family of products is the most widely recognized. Not all OS's are created equal—they run the gamut in terms of functionality, stability, ease of use, and hardware requirements. However, for certain types of powerful computers, Unix is widely regarded as one of the best on the market.[76]

The first version of the Unix operating system was created by two

71. Stallman's views on the necessity of copyleft are not universally accepted in the open source community. However, as the elder statesman of the movement and one of its most outspoken advocates, his influence can hardly be overstated. *See* Stallman*, Why Copyleft, supra* note 65.

72. There is, however, a single Unix specification. *See* THE OPEN GROUP, THE SINGLE UNIX SPECIFICATION, VERSION 3 (2002), http://www.unix.org/version3/.

73. DeLong, *supra* note 8, at 12.

74. *Id.* at 11.

75. THE AMERICAN HERITAGE DICTIONARY (4th ed. 2000), *available at* http://www.bartleby.com/61/34/O0093400.html.

76. *See* Eric Raymond, *Origins and History of Unix, 1969-1995*, *in* THE ART OF UNIX PROGRAMMING (2003), http://library.n0i.net/linux-unix/art-unix-programming/ch02s01.html.

computer scientists at AT&T's Bell Labs in 1969.[77] Unix was not originally envisioned as a large scale project. Bell Labs had recently withdrawn from the consortium designing the MULTICS OS, and AT&T programmers Ken Thompson and Dennis Ritchie, who had grown used to the interactivity that MULTICS offered, sought to create a similar platform to run other projects they were working on.[78] As time went on, Unix became widely used within the Bell Labs programming community and the project eventually developed into a full-fledged OS.

Although originally developed on a DEC PDP-7, in 1973 Unix was completely rewritten in the high-level C programming language, allowing it to be recompiled to run on many different types of hardware.[79] Designing a new OS from scratch is a difficult thing, so the portability of Unix to different hardware configurations made it attractive to many outsiders. Under a 1956 antitrust agreement, AT&T was not allowed to commercialize its non-telephony IP, so thousands of entities were able to obtain Unix licenses practically for free.[80] Proprietary software was not seen as the tremendously valuable asset that it is today, so AT&T distributed the source code as well, sanctioning (and even encouraging) the development of Unix variants such as the Berkeley Software Distribution (BSD) at the University of California.[81]

B. IBM, AIX, and Sequent

By 1984, attitudes towards proprietary software had changed, and AT&T's deregulation allowed it to try to capitalize on its Unix assets. As Unix grew in reputation, large companies like IBM, HP, and Sun became increasingly interested in running it on their own high end machines. This led to the creation of yet more variants, each by a different manufacturer. In 1985, IBM entered into a Unix licensing agreement with AT&T,[82] as well as a sublicensing agreement allowing it to license its Unix derivative called AIX to its customers.[83] IBM could

77. *See id.*
78. *Id.*
79. *Id.*
80. *See* DeLong, *supra* note 8, at 11-12.
81. Portions of BSD code made their way back into AT&T's Unix on at least one occasion. This greatly complicated matters when AT&T set about taking Unix proprietary, while BSD went open source. The resulting lawsuit, which was not settled until 1994, cast a pall of uncertainty over the entire project (and arguably contributed to the acceptance of Linux as an alternative). *See* Wikipedia, Berkeley Software Distribution, http://en.wikipedia.org/wiki/BSD (last visited Feb. 5, 2006).
82. Exhibit A to Second Amended Complaint, SCO Group v. IBM, No. 03-CV-0294 (D. Utah filed Mar. 6, 2003), *available at* http://www.groklaw.net/pdf/Doc-25-A.pdf [hereinafter SOFT-00015].
83. Exhibit B to Second Amended Complaint, SCO Group v. IBM, No. 03-CV-0294 (D. Utah filed Mar. 6, 2003), *available at* http://www.groklaw.net/pdf/Doc-25-B.pdf

now market a complete range of solutions consisting not only of a powerful computer system, but also enterprise level software[84] for them to run.

In September of 1999, IBM merged with Sequent Computer Systems, a company specializing in multi-processor computer design. Sequent had itself purchased a license for Unix and had developed a number of Unix-related software tools and programs (including its own homegrown Unix variant called Dynix/ptx). In the merger, IBM took possession of these assets, and in the process, bound itself to Sequent's license terms, which in some cases varied from its own.

C. The Convoluted History of the SCO Group

1. The Santa-Cruz Operation

Computer manufactures were not the only ones to create Unix variants. Seeing an opportunity to exploit an overlooked niche market, a small software firm named The Santa-Cruz Operation (Old SCO) created a variant of Unix in 1983 that would run on Intel processors.[85] Originally called SCO Unix, but later renamed to OpenServer, Old SCO's variant did not compete directly with the Unix variants developed by the likes of IPM and HP because the Intel processors of the time were still not used for high end computing. Nevertheless, OpenServer was moderately successful and the project gave Old SCO significant experience dealing with Intel-architecture processors.[86]

The emergence of Intel as the dominant processor manufacture in the mid-1990s greatly increased the demand for a server OS capable of running on these processors.[87] Old SCO now found itself in a very

[hereinafter SOFT-00015A].

84. Companies rely heavily on computer systems to run their back office (and often their front office) operations. Downtime can be tremendously costly, so software used in this environment is designed to be highly fault tolerant. The goal of an enterprise class computer system is "five nines" (99.999%) reliability, or an average downtime of less than five and a half minutes per year.

85. The SCO Group, Inc., *History of the SCO Group*, http://www.caldera.com/company/history.html (last visited Feb. 5, 2006).

86. Second Amended Complaint at 13, SCO Group, Inc. v. IBM, No. 03-CV-0294 (D. Utah filed Mar. 6, 2003), *available at* http://www.groklaw.net/pdf/Doc-25.pdf [hereinafter Second Amended Complaint].

87. The 1990s witnessed the explosion of the market for desktop PCs, transforming Intel into a household name. Not content to stay confined to this area of the market, Intel began pouring its resources into improving its processor designs with the hope of expanding into the market for server processors as well. Over time, it increased the computing power of the x86 architecture until it was on par with high end processors. At the same time, it was able to maintain its cost advantage over these processors, for the first time making enterprise computers built with x86 processors an attractive choice. Intel's high price/performance ratio even convinced some manufacturers like HP and Compaq to abandon their own specialized

advantageous position, as their long history of working with Intel meant that they already had a mature Unix platform to deliver to customers. OpenServer offered the reliability of Unix on low cost (and now greatly improved) Intel processors.[88]

By comparison, other Unix variants were now much less attractive because they would only run on processors that were higher-priced, but which offered no better performance. IBM, perhaps sensing that the tide was shifting away from Unix running on its own brand of processors, entered into a joint-development program with Old SCO dubbed "Project Monterey" to design a Unix variant for Intel's next-generation processors.[89] This gave IBM access to Old SCO's copyrighted IP that allowed OpenServer to run on Intel.[90] Project Monterey was never completed and IBM pulled out of the agreement in May of 2001.[91]

2. Novell, Caldera, and the SCO Group

In 1990, AT&T spun off its Unix business into a wholly owned subsidiary called Unix System Laboratories. Then, in 1993, a software company named Novell purchased the subsidiary, along with the Unix code and copyrights. In 1994, several Novell programmers left the company to form the startup Caldera, and in 1995 Novell sold the rights to Unix and UnixWare to Old SCO.[92]

Caldera spent several years distributing a version of the Linux OS before purchasing a number of Old SCO's assets in 2001. These assets included Old SCO's family of OpenServer products, as well as the rights to Unix that Old SCO had purchased from Novell. Finally, in 2002 Caldera changed its name to the SCO Group, giving us the company that we now know as SCO.[93]

D. Linux

In many ways, programming culture of the 1960s and early 1970s mirrored the larger "free-love" spirit of the era. Places like AT&T's Bell Labs and MIT's Artificial Intelligence Lab brimmed with youthful

processor offerings and produce servers based exclusively on Intel chips. *See* Ian Fried, *HP plans to take Alpha to its Omega*, CNET NEWS.COM, Dec. 5, 2002, http://news.com.com/2100-1001-976211.html.

88. *See* Second Amended Complaint, *supra* note 86, at ¶ 48.

89. See Santa Cruz Operation Inc., Quarterly Report (Form 10-Q), at 28 (Feb. 15, 1999), *available at* http://www.groklaw.net/article.php?story=2004030711323697 (last visited Feb. 5, 2006).

90. *See* Second Amended Complaint, *supra* note 86, at 13-14.

91. *Id.* at 14.

92. John C. Dvorak, SCO versus IBM and Linux: Timeline, http://www.dvorak.org/scotimeline/ (last visited Feb. 5, 2006).

93. The SCO Group, Inc., *supra* note 85.

energy and programmers collaborated widely.[94] By the 1980s, however, the communitarianism and pervasive code-sharing of the early days had been largely squelched by proprietary practices. In 1983, a group of programmers unhappy with this situation and led by Richard Stallman (who was particularly disheartened by the decline of the old ethos at MIT's AI Lab) formed the non-profit Free Software Foundation (FSF) to promote the creation of software under the open source model.[95] The Free Software Foundation has made many contributions to the open source movement, one of which is the so-called GNU project to create an open source alternative to Unix.[96]

A functioning OS requires a special software module called a kernel — a piece of code that directly interacts with the hardware. Even though the FSF never succeeded in creating one, a Finnish computer science student name Linus Torvalds was working independently on his own kernel at the same time as the FSF. Dubbed Linux, the kernel could be combined with a number of software components developed by the GNU project to create a fully operational OS.[97] Since its release under the GPL in 1991, the Linux project has received contributions from tens of thousands of programmers and software designers. Over time, Linux has grown into an extremely powerful and fully featured OS, representing to many the best example of what the open source model is able to accomplish.

IV. THE SCO V. IBM LAWSUIT

A. SCO's Allegations

By the end of the millennium, even large for-profit companies had

---

94. Eric Raymond paints a compelling picture of the programmers of the day: "Socially, they were young, exceptionally bright, almost entirely male, dedicated to programming to the point of addiction, and tended to have streaks of stubborn nonconformism — what years later would be called 'geeks'. They, too, tended to be shaggy hippies and hippie-wannabes. They, too, had a vision of computers as community-building devices…. Collaborative development and the sharing of source code was a valued tactic for Unix programmers." *See* Eric Raymond, *Origins and History of the Hackers, 1961-1995*, http://library.n0i.net/linux-unix/art-unix-programming/hackers.html (last visited Feb. 5, 2006).

95. FREE SOFTWARE FOUNDATION, OVERVIEW OF THE GNU PROJECT (2005), http://www.gnu.org/gnu/gnu-history.html.

96. Stallman's decision to go with Unix (as opposed to another OS) was shaped largely by practical considerations: "Unix is not my ideal system, but it is not too bad. The essential features of Unix seem to be good ones, and I think I can fill in what Unix lacks without spoiling them. And a system compatible with Unix would be convenient for many other people to adopt." *See* RICHARD STALLMAN, THE GNU MANIFESTO (2005), http://www.gnu.org/gnu/manifesto.html.

97. *See* Wikipedia, Linux kernel, http://en.wikipedia.org/wiki/Linux_kernel (as of Feb. 5, 2006).

taken notice of Linux. IBM in particular saw advantages in offering this OS to complement its enterprise class servers — it was functionally similar to the various Unix variants that its customers were used to, but could be provided without the costly licensing fees.[98] The cost of the OS typically constitutes a significant portion of the purchase price for an enterprise class server,[99] so IBM could undercut the competition by saying in essence, "buy our hardware and we'll throw in a free OS!" It may have lost some of the revenue it previously generated through AIX licenses, but presumably this was offset by an increase in hardware sales.[100]

In order for IBM's strategy to work, however, significant work had to be done to Linux to put it on par with traditional Unix offerings both in terms of stability and functionality. IBM had, after all, spent nearly 20 years refining its AIX operating system. Customers relied on various versions of Unix to run their "business-critical" applications, so a lower cost would be irrelevant if Linux could not meet the demanding requirements of the enterprise computing environment every bit as well as its proprietary competitors. IBM therefore pledged to help develop Linux by contributing to the project time, programmer talent, and most importantly, certain code assets it possessed.[101] It was these code contributions that formed the basis for the lawsuit from SCO.

As the ostensible owner of Unix copyrights, SCO was concerned that the code IBM was making available to Linux developers rightfully belonged to them. IBM may have had access to it through various license agreements, but these agreements explicitly prohibited any further distribution. In March of 2003, SCO terminated any rights IBM had to Unix and brought suit against them, alleging misappropriation of trade secrets, unfair competition, and breach of contract.[102] In addition, SCO mailed a letter threatening legal action to 1,500 companies using Linux as well.[103] SCO also sued two corporate Linux users (AutoZone and

98. *See* Second Amended Complaint, *supra* note 86, at 17.

99. For example, under IBM's original Unix license, AT&T changed $43,000 for the first CPU and $16,000 for additional CPU the software was run on. AT&T charged an additional $25,000 every time IBM sublicensed Unix (or AIX). Note that these are in 1985 dollars! *See* SOFT-00015, *supra* note 82.

100. SCO alleges a more sinister motivation behind IBM's change of heart. They argue that it was motivated by the company's recent shift away from a licensing revenue model to a service model — that IBM was no longer trying to make money by licensing AIX, but would instead make money from providing services to companies using any variant of Unix or Linux. Distributing Linux as a free replacement for Unix made sense, because it still allowed IBM to sell server hardware, while at the same time making it harder for other companies to make money by licensing their (non-free) versions of Unix. *See* Second Amended Complaint, *supra* note 86, at 19-24.

101. *Id.* at 20-25.

102. *See* Second Amended Complaint, *supra* note 86, at 32-64.

103. Exhibit I to Amended Counterclaims, SCO Group, Inc. v. IBM, No. 03-CV-0294

Daimler-Chrysler) for injunctive relief and damages,[104] presumably to make an example out of them.

SCO's complaint against IBM has since been amended twice, dropping the misappropriation of trade secrets cause of action and adding a copyright claim.[105] It is the copyright claim that is the focus of the remainder of this section. If SCO's allegations prove true, every version of the Linux kernel distributed since roughly the year 2000 is in violation of copyright, as is everyone possessing and using such a copy. However, despite the fact that the trial has now been going on for over two years, SCO has yet to produce the offending lines of code to substantiate its claims.

B. IBM's License Agreements

Recall that IBM originally acquired its rights to Unix through a license from AT&T, and that it later created AIX as a derivative work to be distributed under a corresponding sublicense agreement. The contract licensing Unix from AT&T to IBM ("Software Agreement / SOFT-00015"), along with the sublicense agreement ("Software Agreement / SUB-00015A"), are perhaps the most important documents in the case. They set out the terms under which IBM may use Unix, as well as the rights it has to distribute, prepare derivative works, and sublicense AIX. With their acquisition of Sequent, IBM took over Sequent's portfolio of intellectual property assets. Many of these assets, such as Dynix/ptx were also based on code licensed from AT&T. In addition the licenses themselves, SCO and IBM have also filed with the court several amendments and a letter ("1985 Side Letter") modifying their arrangement.[106] I have reproduced below the most important terms of the various licenses along with a brief explanation of each section.

1. SOFT-00015: IBM's Original Unix License

§ 2.01. AT&T grants to [IBM] a personal, nontransferable and nonexclusive right to use in the United States each [licensed software product (referring to Unix)] identified in the one or more Supplements

(D. Utah filed Mar. 6, 2003), *available at* http://www.groklaw.net/pdf/Doc-41-I.pdf.

104. For more information on these suits, *see* The SCO Group, Inc., *SCO v. AutoZone,* http://www.sco.com/scoip/lawsuits/autozone/index.html (last visited Feb. 5, 2006), *and* The SCO Group, Inc., *SCO v. Daimler Chrysler*, http://www.sco.com/scoip/lawsuits/daimlerchrysler/index.html (last visited Feb. 5, 2006).

105. Second Amended Complaint, *supra* note 86, at 50.

106. Exhibit 15 in Declaration of Jeremy O. Evans in Support of SCO's Memorandum in Opposition to IBM's Motion for Summary Judgment on Breach of Contract Claims, SCO Group, Inc. v. IBM, No. 03-CV-0294 (D. Utah filed Mar. 6, 2003), *available at* http://www.groklaw.net/pdf/Doc-41-D.pdf [hereinafter Side Letter].

hereto, solely for [IBM's] own internal business purposes . . . Such right to use includes the right to modify such [licensed software] and to prepare derivative works . . . provided the resulting materials are treated hereunder as part of the original [licensed products].[107] Although not technically an amendment, the 1985 Side Letter ¶ A2 remarked that: "Regarding Section 2.01, we agree that modifications and derivative works prepared by or for you are owned by you. However, ownership of any portion or portions of [a licensed product] included in any such modification or derivative work remains with us."[108]

Section 2.01 is the basic grant of rights to Unix from AT&T to IBM. These rights include the ability to use Unix for IBM's internal business purposes, and to create derivative works. The side letter provides that portions of derivative works will be owned by their respective authors.

Sections 2.05 and 7.01 make clear that the license applies solely to IBM,[109] and strictly prohibits IBM from transferring any of the licensed assets in whole or in part.[110] However, § 7.06 as amended by the 1985 Side Letter ¶ A9 provides that while IBM may not disclose the licensed assets to anyone,

> [n]othing in this agreement shall prevent [IBM] from developing or marketing products or services employing ideas, concepts, know-how or techniques relating to data processing embodied in [the licensed products] subject to this Agreement, provided that [IBM] shall not copy any code from such [licensed products] into any such product.[111]

Finally, SOFT-00015 sets out the fee structure whereby IBM must pay AT&T based on the number of computers running Unix.[112]

2. SUB-00015A: IBM's AIX Sublicense

IBM is also constrained regarding AIX. As a derivative work based on Unix, AT&T conditioned AIX's distribution on certain terms set out in the sublicensing agreement SUB-00015A. Section 2.01 of the agreement grants IBM the right to furnish third parties with copies of the sublicensed product, so long as those parties agree to certain conditions.[113] In particular, third parties are prohibited from themselves

107. SOFT-00015, *supra* note 82, at 2.

108. Side Letter, *supra* note 106, at 2.

109. SOFT-00015, *supra* note 82, at 3.

110. *Id.* at 4.

111. *Id.*

112. To be more precise, the fees are based on the number of processors being used rather than the number of computers. This distinction is important because large computers often contain multiple processors.

113. SOFT-00015A, *supra* note 83, at 2-3.

redistributing the software. Just as in the license agreement, SUB-00015A also requires IBM to pay a fee for each copy of AIX it distributes.

3. The Sequent License & Sublicense

Sequent, like IBM, licensed Unix from AT&T to create their Dynix/ptx variant. The terms of the license and sublicense were the same as those of IBM, since AT&T used a standard software license form for its products. Nevertheless, there is one crucial difference with regard to the two companies: the 1985 Side Letter between IBM and AT&T made several modifications to SOFT-00015, but there was no corresponding agreement between Sequent and AT&T. Thus, IBM had slightly more expansive rights to Unix than Sequent.

In particular, § 2.01 of AT&T's standard license agreement provides that any derivative works are to be treated as part of the original licensed works. Put another way, derivative works prepared by Sequent were subject to the same restrictions as the original. This adds a new wrinkle to the derivative works doctrine. Ordinarily, the author of an authorized derivative work can do whatever she likes with the new contributions she has made and the original author controls what is done with the original elements.[114] But the Sequent license could be interpreted to mean that AT&T's derivative works authorization is broader than copyright law, conditioned upon AT&T's control of all of Sequent's contributions. Read this way, even elements of Dynix/ptx that are completely original to Sequent would still subject them to the restrictions of the license agreement.

C. Analysis of IBM's Position

1. A Snarled Chain of Title

In 1995, Novell executed a contract filed with the court as the "Asset Purchase Agreement" (APA), in which it sold to Old SCO a number of Unix related assets. The APA is a rather lengthy document that sets out in detail just what was transferred. In § 1.1(a), it spelled out the sale to Old SCO of all "right, title and interest" in the items listed on a form labeled "Schedule 1.1(a)" which was attached to the APA.[115] It also included a list of certain assets *not* transferred, on a form labeled

114. *See* 17 U.S.C. §§ 102, 103 (2006).

115. Exhibit J to Defendant IBM's Answer to the Amended Complaint and Counterclaim-Plaintiff IBM's Counterclaims against SCO, SCO Group, Inc. v. IBM, No. 03-CV-0294 (D. Utah filed Mar. 6, 2003), *available at* http://www.groklaw.net/pdf/Doc-41-J.pdf.

"Schedule 1.1(b)" which was also attached.[116]

Schedule 1.1(a) provided that included in the transferred assets were "[a]ll rights and ownership of Unix," as well as "all technical, design, development, installation, operation and maintenance information concerning UNIX and UnixWare, including source code . . . ."[117] Confusingly, however, Schedule 1.1(b), specifically withheld all copyrights and patents.[118] In 1996, Novell and Old SCO executed an amendment to the APA in a document labeled "Amendment 2." One of the effects of this amendment was to modify Schedule 1.1(b), so that it now excluded:

> All copyrights and trademarks, *except for the copyrights and trademarks owned by Novell as of the date of the Agreement required for SCO to exercise its rights with respect to the acquisition of UNIX* and UnixWare technologies. However, in no event shall Novell be liable to SCO for any claim brought by any third party pertaining to said copyrights and trademarks (emphasis added).[119]

What does this mean for the litigation? The amended APA would seem to say that SCO only owns the copyright to Unix if the copyright is required for SCO to "exercise its rights with respect to the acquisition of Unix."[120] But this, of course, begs the question: is the Unix copyright necessary for SCO to "exercise its rights?" If SCO does not own the copyright, then the entire case is moot.

Novell has, in fact, taken exactly this position. In May of 2003, it publicly asserted that SCO did not posses the copyright to Unix,[121] and, acting as the "true" copyright holder, purported to waive any and all claims regarding IBM's Linux contributions.[122] Immediately thereafter, SCO filed (yet another) suit against Novell for slander of title.[123] This unsettled matter could very well be determinative in the litigation between SCO and IBM, but until the court renders an opinion, the most anyone can do is speculate on the effect all this will have on SCO's claims.

116. *Id.*

117. *Id.*

118. *Id.*

119.Exhibit 29 to [Redacted] Memorandum in Support of SCO's Expedited Motion to Enforce the Court's Amended Scheduling Order Dated June 10, 2004, SCO Group, Inc. v. IBM, No. 03-CV-0294 (D. Utah filed Mar. 6, 2003), *available at* http://sco.tuxrocks.com/Docs/Amendment2.html.

120. *Id.*

121. Second Amended Complaint, *supra* note 86, at ¶ 19.

122. Exhibit K to Amended Counterclaim, SCO Group, Inc. v. IBM, No. 03-CV-0294 (D. Utah filed Mar. 6, 2003), *available at* http://www.groklaw.net/pdf/Doc-41-K.pdf.

123. Complaint at 9, SCO Group, Inc. v. Novell, Inc., No. 04-CV-00139 (D. Utah filed Jan. 20, 2004), *available at* http://www.groklaw.net/pdf/Novell-0.pdf.

2. Amendment X: A Dead-End?

In its briefs to the court, IBM relies on an amendment to SOFT-00015 labeled "Amendment X" to support its claim that it may do whatever it likes with the code licensed from AT&T. This amendment was executed after AT&T sold Unix to Novell. The pertinent portion is § 1, which reads in part:

> [IBM] will have the irrevocable, fully paid-up, perpetual right to exercise all of its rights under the Related Agreements . . . Notwithstanding the above, the irrevocable nature of the above rights will in no way be construed to limit Novell's or SCO's rights to enjoin or otherwise prohibit IBM from violating any and all of Novell's or SCO's rights under this Amendment No. X, the Related Agreements, or under general patent, copyright, or trademark law.[124]

IBM's filings with the court have put particular emphasis on their "irrevocable, fully paid up, [and] perpetual" rights to Unix.[125] They claim that SCO may not terminate their rights under the license agreement because those rights are "irrevocable." However, I contend that IBM's emphasis on Amendment X does little to strengthen its defenses against SCO's allegations.

Amendment X does not expand any of IBM's rights under the license agreement and merely says that IBM's Unix license is no longer terminable by the copyright holder. Notwithstanding the perpetual and irrevocable nature of the amended license, IBM is still bound by its terms. If SCO is able to prove that IBM failed to abide by the license terms, such as those that prohibit revealing confidential source code to others, then IBM is in breach of contract. A license is a contract, so a party in breach of that contract may lose the protections it provides.[126] Without the protection of the license, further use of Unix-related IP would be in violation of copyright law. Since the second sentence of Amendment X § 1 says that SCO may prohibit IBM from violating its copyrights, IBM's reliance on Amendment X is misplaced.

A creative response to this argument might go something like this: even if IBM was in breach of contract, it can never actually lose its license since that license is irrevocable. This would render SCO's

124. Exhibit G to Defendant IBM's Answer to the Amended Complaint and Counterclaim-Plaintiff IBM's Counterclaims against SCO, SCO Group, Inc. v. IBM, No. 03-CV-0294 (D. Utah filed Mar. 6, 2003), *available at* http://www.groklaw.net/pdf/Doc-41-G.pdf.

125. Answer to Amended Complaint and Counterclaim, SCO Group, Inc. v. IBM, No. 03-CV-0294 (D. Utah filed Mar. 6, 2003), *available at* http://www.groklaw.net/pdf/Doc-27.pdf.

126. *See* NIMMER, *supra* note 14, at § 10.15.

copyright claim moot because no matter how much it violated the license terms, it would still technically be covered by it. Since the license was always in force, copyright prohibitions would never be triggered.

However, this counter-argument fails for two reasons. First, it violates the spirit of the agreement by allowing IBM to breach its contract with impunity while leaving SCO without a remedy. Second, it is possible for IBM to be in copyright violation even without a direct breach of the license terms. Remember that a license only grants the rights to engage in a narrow spectrum of activities ordinarily prohibited by copyright. IBM is in violation if it acts outside the scope of these rights. The license certainly did not grant IBM the right to make unrestricted distributions of the Unix source code, so there is no way IBM could legally do so, perpetual license or not.

Careful analysis shows that Amendment X does not advance IBM's case despite its strong language. In the end, the legality of the Linux distributions turns on the nature of the code: if it is original to AT&T then it is subject to copyright, but if it was created in-house then IBM is off the hook.

3. A Better Response: IBM's Right to Create Derivative Works

IBM was granted the right to create derivative works for its own use by § 2.01 of the SOFT-00015 contract. The 1985 Side Letter further states that "we agree that modifications and derivative works prepared by or for you are owned by you. However, ownership of any portion or portions of [the licensed software products] included in any such modification or derivative work remains with us."[127] This parallels standard copyright law — each author is granted copyright in their respective contributions to a collective or derivative work.[128]

I believe that IBM's right to create derivative works is their salvation. As explained in Part II(A) of this paper, there are two sets of rights present in a derivative work: those of the original author and those of the new author. Assuming the new author has permission from the original author to create a derivative work (which the original author is of course free to withhold), the new author will hold copyright in their independent contributions to the derivative. As for the original author, they will retain copyright in the original elements still present in the derivative, *but not the new elements*.[129] The trick is separating the new elements from the old elements. Software code is highly modular, and it

127. Side Letter, *supra* note 106, at ¶ A2.

128. 17 U.S.C. §§ 102, 103 (2004).

129. 17 USC § 103(b) (2004) reads, in part: "The copyright in a compilation or derivative work extends only to the material contributed by the author of such work . . . ."

is possible that code originally written as a derivative could be useful as a standalone product with little or no modification. Since IBM only contributed certain sections of AIX to Linux developers, it is entirely possible that these sections contain only elements which they themselves created independent of the SCO code.[130] If so, they are completely within their rights to do so — SCO has no claim over these elements. Independent creation and modularity are key here. The code may have been a derivative in the sense that it interacted heavily with SCO code (much like a program would interact with a library), but in fact was created entirely at IBM. Even though this code was originally designed to work with code belonging to SCO, large portions might be modular enough so as to be functional with non-SCO code as well. If it turns out that IBM's contributions to Linux consisted entirely of independent, original, and modular code, then IBM has not breached its license terms. No breach of contract means no copyright violation, and SCO's case falls apart.

There are two potential hurdles to this defense. The first lies in the terms of the sublicensing agreement SUB-00015A, while the second has to do with the problem of Dynix/ptx contributions. As detailed in Sections IV(B)(1) and (2), SUB-00015A § 1.04 speaks about derivative works in a slightly different way than SOFT-0015A does. It defines the sublicensed product as computer programs "based" on software products licensed to IBM in SOFT-00015. This ambiguity is important, since AIX is certainly "based" on Unix. As we have seen, IBM-created code, when linked up with Unix code, constitutes a derivative work. Assuming the code is sufficiently modular to be decoupled from Unix code, IBM can argue that once decoupled, it is no longer derivative and IBM is free under the 1985 Side Letter to do with it as it wishes. If, on the other hand, the term "based" is substituted for "derivative," this decoupling is more difficult. Even decoupled, IBM's code is arguably still "based" on Unix, in the sense that linking with Unix was the original reason for its creation. The proper interpretation of these two terms is something that only a court can determine, but it is not readily apparent why a court would opt to read the term "based" more broadly than the term "derivative."

More troubling are the Sequent licensing terms set out in Section IV(B)(3), since arguably they *do* apply to any and all products based on the original version of Unix. A number of the items that SCO has

130. Defining the scope of derivative works in software is difficult and controversial. Some would argue that these additions are not derivative at all, but rather, completely original pieces of work. *See* Mitchell L. Stoltz, *The Penguin Paradox*, 85 B.U. L. Rev. 1439, 1441, 1449-51 (2005); Greg R. Vetter, *"Infectious" Open Source Software: Spreading Incentives or Promoting Resistance?*, 36 RUTGERS L.J. 53, 94-110 (2005).

alleged IBM contributed are software components originally developed by Sequent for Dynix/ptx under its own Unix license. Without knowing the actual lines of code in question, it is impossible to tell whether Linux does in fact contain elements of Dynix/ptx. If it does, then on its face, the distribution of these elements is in violation of the license.

This eventuality seems unlikely however. It seems unlikely the Unix derivative works right IBM negotiated in the SOFT-00015 and SUB-00015A licenses would be limited because of IBM's future purchase of the more restrictive Unix derivative works right in the Sequent license. In other words, the Unix derivative works rights of the SOFT-00015 and SUB-00015A licenses should supersede the more restrictive Unix derivative works right of the Sequent license. If not, the liability is still reduced as only Sequent-specific derived works suffer from this liability, so SCO will need to point these out.

V. WHAT THIS MEANS FOR F/OSS DEVELOPMENT

Having seen an instance of litigation involving the two models of software development, what does this portend for the future of open source development (and Linux in particular)? I argue that while the Linux project is secure for the time being, the long term health of the model requires that F/OSS projects be seen as viable alternatives to their proprietary counterparts. To achieve this, an aggressive deployment of the GPL ensures that at the very least F/OSS projects do not start off at a disadvantage, while at the same time preserving the incentive structure that has served the model so well thus far.

A. *Implications for Linux and Other F/OSS Projects*

Assuming that SCO produces specific lines of code that the court finds Linux infringes upon, the immediate effect will probably be quite small. Within the world of open source development, the Linux programming community is one of the largest and most active.[131] The offending code would eventually be rewritten independent from access to SCO's code, leaving the software immune from further intellectual property claims (at least by SCO).

IBM itself would be in hot water; SCO's complaint has asked for damages in excess of two billion dollars. But more damaging to Linux, and the open source movement as a whole, would be the effect an

131. While determining the exact number of contributors to a large F/OSS development project like Linux is difficult, a 2001 analysis of the Red Hat 7.1 Linux distribution revealed that it contained over 30 million lines of source code. The study estimated that under the proprietary model, the amount of programmer time this represented would be valued in excess of $1 billion. David A. Wheeler, More Than a Gigabuck: Estimating GNU/Linux's Size, http://www.dwheeler.com/sloc/ (last visited Mar. 14, 2006).

adverse decision would have on users — corporate users in particular. If Linux is indeed held to be an infringing work, each act of copying is an instance of copyright infringement. While SCO did not specify any specific legal action it was contemplating in the 1,500 letters it sent out, the penalties for companies using Linux could potentially be severe.

The SCO litigation raises the very real danger that users will be dissuaded from using Linux solely out of fear.[132] Even if the open source community were to produce a "clean room" version that could be verified from top to bottom as being free from proprietary code, this would take time to develop. In the interim, a company that wanted to be absolutely certain that it would not be liable for copyright infringement would have to migrate off the Linux platform. Migrating an entire company's computer infrastructure to a new operating system is extremely difficult, costly, and time consuming. Once the migration is complete, there is little incentive to migrate back to Linux. Few would be willing to risk getting "burned" again from some other challenge to open source.

Luckily, these dangers have not materialized as of yet; companies do not seem to be taking action one way or the other with regard to Linux. The suit against Daimler-Chrysler was dismissed,[133] while AutoZone has vigorously defended itself and the litigation has bogged down. With the IBM trial is currently scheduled for 2007, it may be some time before there are further developments.

Even if the Linux project is eventually vindicated, there is nothing to prevent this scenario from being repeated in the context of some other F/OSS project. Unless a developer writes all of their code from scratch, there is always the danger that their program will somehow be "tainted" by the presence of unauthorized code. This threat will probably never be fully neutralized given the cost of writing good code and testing it thoroughly and the incentive this creates for developers to reuse code.

When it comes to detecting the presence of unauthorized code, proprietary developers possess an informational advantage. Open source code is available for all to see, allowing proprietary developers to inspect it for infringement. Conversely, the secret nature of proprietary code means that an open source developer may not know that they are in

132. The practice of spreading FUD (fear, uncertainty, doubt) about Linux has a long and storied history among its competitors. *See* Eric Raymond, The Halloween Documents, http://www.catb.org/~esr/halloween/index.html (last visited Mar. 23, 2006). Professor McGowan believes that the current litigation is part of a larger rhetorical battle being waged against the open source model, and that this battle will be decided largely without regard to the legal merits of the various claims being made. David McGowan, *SCO What?* (Univ. of Minn. Law Sch. Legal Studies Research Paper Series, No. 04-9, June 6, 2004), *available at* http://ssrn.com/abstract=555851.

133. SCO Group, Inc. v. DaimlerChrysler Corp., No. 260036 (Mich. Ct. App. Jan. 31, 2005), *available at* http://www.groklaw.net/pdf/DC-8.pdf.

possession of infringing proprietary code until it is too late.

It is often unclear how code finds its way out of a proprietary product. Perhaps a lowly programmer was particularly proud of it and posted it to an internet bulletin board without the consent of their employer. Perhaps it was copied out of a derivative work by a licensee of the derivative's author, who mistakenly believed that they possessed the proper license. Or perhaps it was ripped straight out of a copyrighted work by a large corporation and illegally donated to an open source developer — as SCO alleges occurred at IBM.

Going forward, developers would be well advised to avoid code of uncertain origin. But what about F/OSS programs created before the current age of heightened awareness? No developer wants their first notification of infringement to be through service of process. Unfortunately, there is no easy way to determine software pedigree ahead of time.

B. Long Term Viability

Success breeds success. In the F/OSS context, the more widely a program is used, the larger the pool of potential contributors. For programmers, there are psychological rewards associated with being a part of a successful project, such as increased standing in the eyes of other programmers and a heightened sense of accomplishment in a job well done. Working on a credible alternative to a proprietary product also awakens a natural sense of competition and instills F/OSS contributors with a purpose — beat Microsoft! — that working on a hopeless also-ran does not. People strive harder when the race is close.

F/OSS projects have flourished in part because they provide a creative outlet for the participants; indeed the open source movement is an outgrowth of this previously underserved need. Yet network effects which are so profoundly at work in this environment can operate in a negative fashion too. Just as success breeds success, a loss in momentum breeds attrition. The model is heavily dependant on non-monetary rewards to motivate contributions, so anything that interferes with that incentive system risks alienating a large segment of participants. When the Linux suit was first announced, what struck fear into the hearts of open source advocates was not that it threatened to make the software permanently unusable, but that the psychic injury to contributors that a SCO victory would have caused might have been irreparable to the project's continued existence.

The GPL plays an important role in preserving the continued vitality of F/OSS projects. First, it ensures a level playing field during development, increasing these projects' chances of being credible alternatives to proprietary software. Given that proprietary developers do not want F/OSS developers to be able to free ride off of their work (and will go to court to prevent this), basic fairness dictates that they should not be allowed to appropriate F/OSS code for themselves. Open source licenses without copyleft provisions do nothing to prevent this eventuality; only the GPL can prevent a proprietary developer from incorporating open source code into his own project.[134] This is probably less of an issue in "clash of the titans" match-ups such as Linux versus Windows; Microsoft has more than enough resources to develop its own code without needing to "mooch" off F/OSS developers. However, not all programs are Microsoft Windows, and for smaller software projects, it can be difficult for an open source developer to get ahead if his new ideas are constantly in danger of being co-opted by proprietary "competitors."

Second, by preventing this unwelcome appropriation, the GPL preserves the open source model's incentive system. To quote Harvard Law professor Jonathan Zittrain, the self-perpetuating aspects of copyleft can be seen "as a quid pro quo for using and improving upon those works, to compel others to contribute to that pool any improvements they make and wish to release. . . . [If those works] stood to be proprietized by some future party, current contributors might be tempted to hold back their contributions to the common project."[135]

Proprietization of F/OSS code subverts the purpose of the entire open source movement, and particularly affects those who strongly identify with the tenets "free software." The open source movement can ill-afford to lose this important segment of its membership.

CONCLUSION

The open source model is a welcome alternative to the proprietary model of software development. While each possesses its own set of strengths and weaknesses, choice is rarely a bad thing. Because there is much interest in the long term viability of F/OSS projects, SCO's suit against IBM has generated a great deal of consternation among Linux users. Luckily for these users (and open source developers in general), I

134. Even a programmer who is not philosophically opposed to keeping code secret—or who does not mind seeing someone else take their free code and incorporate it into a proprietary work—is still likely to favor measures that keep the playing field level because this gives open source alternatives the best chance of coming out on top (thereby "sticking it to" proprietary naysayers). As Eric Raymond puts it: "The typical pragmatist attitude is only moderately anticommercial, and its major grievance against the corporate world is not 'hoarding' per se. Rather it is that world's perverse refusal to adopt superior approaches incorporating Unix and open standards and open-source software. If the pragmatist hates anything, it is less likely to be 'hoarders' in general than the current King Log of the software establishment; formerly IBM, now Microsoft." Raymond, *Homesteading the Noosphere*, *supra* note 50.

135. Jonathan Zittrain, *Normative Principles For Evaluating Free and Proprietary Software*, 71 U. CHI. L. REV. 265, 279 (2004).

believe they have little to fear from this litigation because SCO will struggle in proving IBM did not have the right to contribute its derivative and independent code to Linux. That said, the risk of unauthorized code use is still present, so developers are advised to use caution. More broadly, the future health of the open source model requires that F/OSS programs be seen as legitimate alternatives to proprietary software. By employing innovative strategies like the GPL, F/OSS developers not only ensure that they compete on a level playing field with proprietary developers, but they also preserve the incentive structure necessary to motivate future contributions.