Normative Principles for Evaluating Free and Proprietary Software

Normative Principles for Evaluating Free and Proprietary Software

Jonathan Zittrain

Final Proofs, 01/08/04 2:05:58 PM

Normative Principles for Evaluating Free and Proprietary Software

Jonathan Zittrain†

The production of most mass-market software can be grouped roughly according to free and proprietary development models. These models differ greatly from one another, and their associated licenses tend to insist that new software inherit the characteristics of older software from which it may be derived. Thus the success of one model or another can become self-perpetuating, as older free software is incorporated into later free software and proprietary software is embedded within successive proprietary versions. The competition between the two models is fierce, and the battle between them is no longer simply confined to the market. Claims of improper use of proprietary code within the free GNU/Linux operating system have resulted in multi-billion dollar litigation. This article explains the ways in which free and proprietary software are at odds, and offers a framework by which to assess their value—a prerequisite to determining the extent to which the legal system should take more than a passing, mechanical interest in the doctrinal claims now being pressed against GNU/Linux specifically and free software generally.

INTRODUCTION

For the past twenty years, the modern landscape of information technology has accommodated competing spheres of software production. These spheres can be grouped roughly around two poles warring for dominance in the field. On one side is proprietary software, which typically provides cash-and-carry functionality for the end-user. Its source code "recipe" is nearly always hidden from view as a technical matter, and as a legal matter it cannot be used by independent programmers to develop new software without the rarely-given permission of its unitary rights holder. On the other side is "free" software, the recipes of which are open to public view and use. Some free software is further "copylefted," that is, copyrighted for the purpose of incorporating license restrictions designed to ensure that anyone who uses and releases the copylefted code as a component of new software must *also* release that new software under copyleft's otherwise-permissive terms.[1]

---

† Jack N. and Lillian R. Berkman Assistant Professor for Entrepreneurial Legal Studies, Harvard Law School. I thank participants in this Symposium and the Harvard Law School Summer Research Program, Derek Bambauer, Tom Brown, Noah Eisenkraft, Terry Fisher, Jerry Kang, Douglas Lichtman, Isaac Lidsky, Mark Lemley, John Palfrey, Mark Roe, Steven Shavell, and Eugene Volokh for helpful discussions and criticisms, and Mary Bridges, Megan Kirk and Greg Skidmore for research assistance on this ongoing project.

[1] See Severine Dusollier, *Open Source and Copyright: Authorship Reconsidered?*, 26 Colum J L & Arts 281 (2003) (discussing the development of copylefting); Christian H. Nadan, *Open Source Li-*

The legal forms of proprietary and free software production cannot coexist within a given piece of code. The proprietary form relies on the existence and enforcement of prevailing copyright law. In contrast, copylefted code asserts a thus far legally untested license pegged to copyright in order to establish the restriction that successor code must be licensed in precisely the same way, namely with its source code freely available. The incommensurability of free and proprietary legal structures within a single piece of software is but one facet of a fundamental philosophical divide between the structures' respective hard-core adherents. Many who promote copylefted software do so only as a second best to the politically unattainable eradication of nearly all existing available proprietary rights in software. Sellers of proprietary software, on the other hand, believe that the growing popularity of copylefted software threatens both their individual business models and the PC software industry generally—possibly leading to an equilibrium in which later innovation is largely forestalled, since no one can readily monopolize derivatives to popular free software to recoup large investments in improving the original works.

The clash between models is currently unfolding most vividly among developers of microcomputer operating systems. This is because there are wildly popular operating systems generated by both models, some of which are functionally quite similar (such as the free GNU/Linux on the one hand and various proprietary Unixes on the other), others of which are largely incompatible and require path-dependent commitments by consumers to one or the other (such as the GNU/Linux on the one hand and the Microsoft Windows family on the other). Such intense competition provides extra incentives for one side to advance or fund claims of legal impropriety against the other, and the stakes in competition over operating system adoption are particularly high: horizontal network effects mean that widely adopted operating systems can snowball into even further adoption, and successful operating system makers can seek advantages in the marketing and sale of vertically related applications or hardware tied to their operating systems.

The ongoing case of *SCO Group v International Business Machines Inc*[2] represents a major legal battle between the spheres of free and proprietary software, at a moment when dominance over operating systems for Internet server-related purposes is truly up for grabs. Through a rather complex chain of title, SCO has become the repository of a number of exclusive intellectual property rights surrounding the original Unix operating system initiated in 1969 at Bell Labs. The GNU/Linux operating system was self-consciously developed by others beginning in 1984 to be functionally similar to Unix but completely nonproprietary; it was written with "fresh" code,

---

censing: Virus or Virtue?, 10 Tex Intel Prop L J 349 (2002) (emphasizing the importance of programmers understanding open source licensing).

[2] Docket No 2:03CV00294 (D Utah), available at http://www.utd.uscourts.gov/documents/ibm.html (visited Dec 16, 2003).

so that none of its source code recipe could be said to be copied or inherited from Unix. In the late 1990s a number of firms saw strategic benefit to making contributions to the GNU/Linux code base, even though the understanding was that such contributions could not be proprietized by the contributing firms. The case began when SCO claimed that IBM had impermissibly contributed code to GNU/Linux that in fact did come from Unix (specifically, IBM's own licensed proprietary variant of Unix called AIX), thus violating IBM's agreement with SCO and "poisoning" GNU/Linux in ways that implicate state unfair competition and trade secret law. SCO has announced it will be adding claims of copyright infringement as well.[3] SCO is seeking billions of dollars in damages from IBM and has terminated IBM's license to sell AIX (which IBM claims SCO cannot do). Exactly what code is alleged to have been stolen has not yet been made public, and in the meantime SCO has asked, under general threat of litigation, that every entity making use of GNU/Linux pay hundreds of dollars in licensing fees to SCO, the terms of which pretermit modifications or redistribution of GNU/Linux source code.[4]To the extent they are heeded, SCO's demands would result in the conversion of the flagship example of free, copylefted software into a proprietary system owned and controlled by SCO, one user at a time.

The facts of *SCO v IBM*—many of which inhere in the intricacies of decades-old licensing agreements and the similarities between particular lines of code—will be developed over a trial process that could take years. I am concerned less with the facts underlying a doctrinal claim by SCO against IBM than I am with the implications of the case for the overall conflict between free and proprietary software. This is because the open development model for free software invites continuing litigation over intellectual property rights. Such litigation could be initiated by any number of firms that can plausibly claim infringement, for reasons that may include a larger strategic targeting of the free software development model. To properly assess and remedy these claims, the legal system must have a framework with which to judge the social value of free software's open development model—and thereby decide how to apply ambiguous doctrine when claims of infringement arise, including claims that purport to reach not just developers but also consumers of that software. This is especially so because nonproprietary software doesn't simply encompass discrete code bases like that of GNU/Linux: the fundamental protocols of the Internet and its subset World Wide Web are themselves nonproprietary and potentially vulnerable to claims of intellectual property ownership by any number of parties.

---

[3] See http://enterprise-linux-it.newsfactor.com/story.xhtml?story_title=IBM_Wins_Skirmish_in_SCO_Battle&story_id=22813 (visited Jan 8, 2004).

[4] See http://www.sco.com/scosource/description.html (visited Jan 8, 2004).

In order to normatively evaluate the spheres of free and proprietary software, it is helpful to briefly review their defining characteristics. This becomes especially important due to the still-ongoing semantic debates among software developers about what counts as "free" and "proprietary."[5] In reality, however, these debates often stand in for differing ideological preferences about how software should be written or shared.

I. DEFINING FREE AND PROPRIETARY SOFTWARE ALONG THREE DIMENSIONS

A. Legal Differences between Free and Proprietary Software

In 1984 Richard Stallman quit his job at the MIT artificial intelligence lab to develop what he called "free" software—software that others could copy and change as they pleased. He found this type of sharing ethically important and endeavored to rewrite the proprietary Unix operating system from scratch so that his version would be substitutable for Unix without infringing any copyrights in the existing Unix code. He named his project GNU, for "GNU's Not Unix." GNU culminated in 1992 after the contribution of a small but crucial piece of code captained by computer science student Linus Torvalds. Torvalds's addition of the "Linux kernel" made GNU (now called GNU/Linux, or, confusingly, just Linux) complete.[6]

Stallman's own vision of free software evolved from that of software released without authorial restrictions on copying or derivation—a notion that could be accomplished by simply releasing one's work into the public domain—into software governed by a licensing scheme that would prohibit authors of derivations from placing restrictions on the distribution of their derived works that had not been placed on the distribution of the original code. Preventing the "proprietization" of derivative software lies at the heart of Stallman's "copyleft" General Public License (GPL), under which GNU/Linux and a great deal of other free software are now distributed.[7] Copyleft was styled as a form of legal jujitsu—a use of copyright (and the availability of accompanying licensing terms) to "protect" free software from being more restrictively copyrighted by those who added their own code to the existing software and redistributed the end product. Indeed, copyleft's restrictions may kick in even before a later work incorporates sufficient code from a copylefted program to be considered a derivative

---

5 See Nadan, 10 Tex Intel Prop L J at 351–55 (cited in note 1) (distinguishing among the "free," "open source," and "proprietary" software movements).

6 See Richard Stallman, *The GNU Project*, online at http://www.gnu.org/gnu/thegnuproject.html (visited Dec 16, 2003).

7 See GNU General Public License, Terms and Conditions for Copying, Distribution and Modification, online at http://www.gnu.org/copyleft/gpl.html#SEC3 (visited Dec 16, 2003).

work; Stallman's license is written to cover any work "that in whole or in part contains or is derived from the Program or any part thereof."[8]

Stallman's flagship GPL has been joined by a flotilla of other similar licenses by other authors, all with their own variations. Beyond the universal trait of allowing others to build upon the base code and release the result, some, such as the license for a variant of Unix called BSD, allow others to build upon the underlying software without passing on the accompanying "copyleft" restrictions. The BSD license materially differs from a wholly public domain release only in that it requires a particular kind of credit or attribution for the original author on whose work the new program is based. Such works are usually called "open" rather than "free," or if "free" are qualified as "but not copyleft."[9] Other licenses allow new derivative works only under some form of copyleft restriction, but vary on whether "nearby works" (that is, works bundled on the same CD-ROM or linked to but not literally incorporated into the licensed code) must themselves be copylefted.

Proprietary software in mass distribution almost uniformly reserves all rights to the author except a license to "run" the software on the purchaser's computer.[10] For many users this restriction does not bar desired activity; non-programmers will evaluate a software purchase on the basis of the program's functionality rather than on its use as a base in producing other software. The most popular software includes tools that allow a user to adjust how the software operates in fine detail. Far beyond checking a box in a "Preferences" window, powerful "macro" languages are often built in to allow skilled users to alter the way in which their proprietary software operates, essentially writing software that is run by their own software. There has been no suggestion that macros written by users cannot themselves be

---

8 Id § 2(b). Compare 17 USC § 101 (2000):

> A "derivative work" is a work based upon one or more preexisting works, such as a[n] . . . 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 which, as a whole, represent an original work of authorship, is a "derivative work."

9 For an introduction to open source software, see Nadan, 10 Tex Intel Prop L J at 350 (cited in note 1).

10 Software is subject to copyright, and new software incorporating elements of old software can comprise a derivative work. See 17 USC § 117 (2000) (delineating limited exceptions to exclusive rights for computer programs); Dun & Bradstreet Software v. Grace Consulting 307 F.3d 197, 212 (3rd Cir. 2002) (cert denied 2003) ("Grace's W-2 program using Copy and Call commands copies Geac's computer copyrighted code. Thus, it is a derivative work; the inclusion of the Copy and Call commands makes Grace's W-2 programs infringing, derivative works of Geac's copyrighted software."); Sega v. Accolade, 977 F.2d 1510, 1518-19 (1992) (Intermediate copying of computer object code may infringe exclusive rights granted to copyright owner to reproduce work, prepare derivative works based upon copyrighted work, and to authorize preparation of copies and derivative works, regardless of whether the end product of the copying also infringes those rights).

transferred, copied, or licensed however the users desire, without claim of right by the proprietary software maker. (The absence of any claim to users' macros may be more a market decision than a presumption that such a claim would not be sustained.)

For proprietary software not in the mass market, any number of arrangements might be agreed upon between the vendor and the consumer. The user might be permitted, for example, to see the source code and make changes, but not to distribute those changes to others. (Without such permission, consumer changes, if substantial enough, would comprise derivative works—the creation of which is a right reserved to the original author.)[11] Most recently, in an apparent response to the successes of the free software movement, certain proprietary software makers have attempted to allow approved users some measure of ability to adapt proprietary software to their own uses by accessing and altering the software's source code. Microsoft's "Shared Source Initiative" (SSI) is one such example.[12] Through it, certain users can adapt Microsoft code to their own special needs, so long as they promise not to further share or sell that code to others. It is too early in the deployment of SSI to gauge how central it is to Microsoft's software marketing efforts. While the company has not released systematic data on its adoption, Microsoft's manager of the SSI reports that over 650,000 developers are using shared source code from the company.[13]

B. Technical Differences between Free and Proprietary Software

As suggested by the use of the term "recipe" for the code underlying functioning software, a given piece of software typically exists in two related components: source code and object code. Source code is what programmers write; object code is what computers run. Software developers produce object code from source code through the use of a compiler. Object code without source code is useful for running a program, but not for easily learning how it works or was written. An attempt to "decompile" object code back into source code yields instructions that bear little resemblance to the original recipe for the program, even if they are functionally equivalent. To analogize, imagine a "decompiled" recipe that calls for adding ¾ teaspoon of sugar, mixing, and then removing ¼ teaspoon of sugar. This is perhaps functionally equivalent to the original recipe that calls for adding ½ teaspoon of sugar, but would be a much more frustrating, though to be sure not impossible, task.

Free software—at least as defined by Richard Stallman—is not free if the source code is not also included with the object code. Stallman has fa-

---

[11] See 17 USC § 106(2) (2000).

[12] See http://www.microsoft.com/resources/sharedsource/Initiative/Initiative.mspx (visited Dec 16, 2003).

[13] Email from Jason Matusow to Jonathan Zittrain, Dec 1, 2003 (on file with author).

mously stated that the "free" in "free software" is more like free speech than free beer.[14] One can charge for a particular copy of free software so long as the source is provided and a specific bundle of rights is delivered with the software—such as the right to further copy it.[15] On the other hand, one can give away proprietary software, like the Internet Explorer Web browser, without charge, and it still isn't "free." The GPL requires that any release of a covered program (for example, one whose author drew upon someone else's GPL'd code to write it) must also readily make available the program's corresponding source code.

Releasing the object code without the source code has been a hallmark of proprietary software, complementing the creator's exercise of a legal right to prevent the use of source code in new works with a *technical* barrier to unauthorized use. Still, these legal and technical facts are analytically independent of one another. Some proprietary programming code happens not to be compiled, so the "executable" and source are one and the same thing. For example, the code by which Web pages are rendered is typically this way; anyone viewing a Web page can, in most browsers, ask to "view source" and will promptly be shown the code by which the page came about. To copy such code and use it for purposes other than viewing the Web page in question might well infringe the copyright of the code's author. Apart from the more typical instances in which the technical nature of coding permits object code to be given to users without easily revealing to them its ancestral source, phenomena like Microsoft's SSI illustrate that vendors might willingly forgo the technical protections of releasing object code without source code, while still asserting strong legal protection over the uses of that released source code.[16]

Conversely, one might imagine a software author releasing, for whatever reason, only object code into the public domain. Such a release would leave no legal rights vested in the author, but the author might still be in an advantageous position to further exploit the software in new works, since only the author would have access to the clean source code for the program.

Thus, the legal and technical protections afforded respectively by copyright in a program and the selective release of its object but not source code are by no means coextensive. In the technical realm, one can only make the general observation that there currently exists a spectrum—from public domain, through free, on to shared and then "fully" proprietary—and

---

[14] See http://www.gnu.org/philosophy/free-sw.html (visited Jan 7, 2003).

[15] See GNU Project, *The Free Software Definition*, online at http://www.gnu.org/philosophy/free-sw.html (visited Dec 16, 2003) (noting that free software requires giving users freedom to run, copy, distribute, study, change, and improve the software).

[16] Source code as well as object code receives copyright protection. See, e.g., *Computer Associates International, Inc v Altai, Inc*, 982 F2d 693, 702 (2d Cir 1992). In addition, The Sixth Circuit has held that source code is a form of speech under certain circumstances and thus subject to First Amendment protections. *Junger v Daley*, 209 F3d 481, 485 (6th Cir 2000).

that along that spectrum one tends to see increasing legal and technical restrictions on code's use.

C. Developmental Differences between Free and Proprietary Software

A third analytically independent difference along the spectrum from free to proprietary is the manner in which the software is typically developed. As most famously chronicled in Eric Raymond's essay *The Cathedral and the Bazaar*,[17] at least two prevailing models of software development exist, roughly grouped around notions of free and proprietary software. The former "open" mode of development grows out of venerable academic computer science and amateur tinkering circles (amateur not in the sense of dilettante, but rather in the sense of one who undertakes something more out of love or fascination than professional duty). Open development emphasizes collaborative work even among strangers and across or even entirely outside of the boundaries of firms, with users of a piece of software themselves contributing changes and improvements to the larger project over time. These improvements are often not made in response to others' requests, but rather in order to make the software more usable to the person making the changes. The perceived absence of legal or technical constraints on the use and modification of a program's code can, under the right circumstances, spawn hundreds of variants among myriad developers. For example, the operating system Unix counts dozens of variants in its line, some licensed from an upstream claimant to proprietary rights.[18] Some versions of Unix available in source code to the tinkering public (such as FreeBSD and the self-consciously intended-as-free GNU/Linux) have been thought by those working on them to be free of practically all legal restrictions.

The tremors of the *SCO v IBM* lawsuit—in which a single company has claimed rights to pieces of Unix that are claimed to have been incorporated into GNU/Linux—are significant precisely because they undermine the collaborative development model. Verifying the "legality" of code offered by a contributor to a project is both superfluous to its technical merit—and thus possibly only of peripheral interest to project leaders—and nearly impossible to do with any thoroughness, since the proprietary code that is, by hypothesis, the source of pilfered free code is not accessible to project leaders assembling the free code from willing sources purporting to have the right to offer it. Any doubt as to the legal character of the result of the collaborative development process, if taken seriously, could impel software developers to work only on code that they themselves originated, or

---

[17] Eric S. Raymond, *The Cathedral and the Bazaar*, 3 First Monday (Mar 2, 1998), online at *http://www.firstmonday.dk/issues/issue3_3/raymond/#d1* (visited Dec 16, 2003) (arguing that open source software may win out because, inter alia, it has a larger talent pool to draw on than do commercial companies).

[18] See Éric Lévénez, *Unix History*, online at http://www.levenez.com/unix (visited Dec 16, 2003) (providing a chart and table listing UNIX derivatives).

for which ownership interests are as clear as possible, with clarity achieved through certification procedures among contributors that circumscribe the number of participants and the pace of their contributions.

The typical mode of development for proprietary software, by contrast, revolves around a firm and those software developers or other firms in specific privity to it. Software is conceived of, written, and tested in-house, and the firm takes some responsibility for user support and upgrades, all typically in response to market pressures and influences. More important for legal purposes, a proprietary firm can stand behind the pedigree of its code, both because it presumably originated in controlled and known circumstances and because the absence of accompanying source code makes the firm's offerings difficult to examine for evidence of theft, whether from competing proprietary companies or from copylefted, publicly available software.

II. NORMATIVE PRINCIPLES FOR THE EVALUATION OF SOFTWARE PRODUCTION MODELS

The proprietary model of software development joins commoditized creative counterparts, such as literature, movies, and television, in relying heavily on government-created rights for its business model. This implicates the much-explored question of just what the nature and scope of IP rights in software should be, and the newer, less-explored question of how those rights should play out when in conflict with free software.[19] The latter question arises when assessing the extent to which free software's copyleft licenses should be enforced (something no court has ever squarely ruled on), or, more generally, in choosing sides in cases of "code poisoning," when, as in *SCO v IBM*, the code from one model of development is said to have routinely tainted the code of another.

To create a framework in which one might evaluate lawsuits such as *SCO v IBM*, I will explore several normative goals and see how each mode of software development fares with respect to each one. I conclude there is no one-size-fits-all model; for this and other reasons, the legal system should be chary of resolving ambiguous doctrinal issues in *SCO v IBM* and other potential suits too strongly or mechanically in favor of the proprietary

---

[19] The literature is so far silent on the prospect of free software becoming "poisoned" by proprietary software, but for more general discussions on the differences between open source and proprietary software, see Klaus M. Schmidt and Monika Schnitzer, *Public Subsidies for Open Source? Some Economic Policy Issues of the Software Market*, 16 Harv J L & Tech 473 (2003) (applying modern economic theory to the software market in general and analyzing whether open source software can mitigate the potential market failures); Marcus Maher, *Open Source Software: The Success of an Alternative Intellectual Property Incentive Paradigm*, 10 Fordham Intel Prop, Media, & Enter L J 619 (2000) (employing complexity theory to explain both the success of the open source movement and its potential problems).

model, which for reasons I will explain is more consistently likely to be represented by the plaintiff in such suits.

A. Nonutilitarian Objectives/Moral Rights

The justifications for free software generally (and copyleft specifically) invoked by its originator, Richard Stallman, are based on free software as a social movement. This movement centers on the belief that software (and all nonrivalrous goods) simply *ought* to be free as an ethical matter and that making it so as a matter of policy would not cause software development to grind to a halt.[20] Stallman's view no doubt discounts the mercenary nature of many software coders, but in any case simply asserts the unfairness of only some members of society possessing a benefit—however it originated—that cannot legally be shared, even though there are no logistical or technical barriers to its immediate enjoyment by all. This belief not only applies to access to executable code, but also access to source code, so that others may learn from it and improve upon it.

So long as government policy is willing to permit proprietary claims to one's work (or to one's creative improvement upon public domain works), Stallman believes in a moral right to assert a proprietary claim *against* the proprietization by others of one's own work. Stallman is thus among those who justify intellectual property rights—if at all—not for instrumentalist reasons, but rather due to the belief that one ought to be able to control one's own intellectual fruits because it's simply *right* to be able to do so. For example, Bruce Springsteen objected to Bob Dole playing "Born in the U.S.A." at campaign rallies, though he found sending a widely reported fax to the campaign to be all the redress he wanted.[21] The same moral beliefs that can underlie a desire not to have one's creative work enjoyed without payment or permission – that is, a stance that authors deserve the right to proprietize their works – are the very ones that can bolster Stallman's claim that copyleft is appropriate, regardless of its instrumental effects on innovation or other values, because it effectuates the wishes of the software's creator.

Stallman's outlook contrasts with that of Eric Raymond and his Open Source Initiative,[22] a non-profit marketing effort for free software begun in the wake of Netscape's announcement that its browser would be de-

---

[20] See Richard M. Stallman, *Why Software Should Be Free*, in Joshua Gay, ed, *Free Software, Free Society: Selected Essays of Richard M. Stallman* 119 (Free Software Foundation 2002). See also Richard M. Stallman, *Why "Free Software" Is Better than "Open Source,"* in Gay, ed, *Free Software, Free Society* 55 (describing the differences between free and open source software, and positing that open source is merely a development methodology, while free software is a social movement).

[21] See Shauna Snow, *Morning Report; Arts and Entertainment Reports from the Times, National and International News Services and the Nation's Press*, LA Times F2 (Oct 10, 1996).

[22] See http://www.opensource.org (visited Dec 16, 2003).

proprietized.[23] Raymond eschews moral argument in favor of espousing pragmatic advantages to individual firms that adopt nonproprietary software.[24] The division between Stallman's "free software" movement and Eric Raymond's "open source" movement reflects in large measure a difference in normative outlook—Stallman's deontological, Raymond's consequentialist. Stallman focuses on the innate responsibilities of software authors to share their work with others (even if they charge per physical copy of the work); Raymond focuses on the benefits that accrue to authors and users if they avail themselves of a collaborative development model and a sharing of source code.

If authorial control is the end, then copylefted software and proprietary software can be reconciled – they merely reflect different desires by authors, each exercising control over respective works. The legal framework could recognize strong proprietary rights, to include the enforcement of copyleft licenses where authors elected to use them. If sharing existing works is the goal (rather than a means to a more general maximization of social welfare that would take other factors into account), then proprietary rights should be minimized and copyleft is unnecessary, since there is then no potential proprietization of one's work to forestall through "jujitsu" licensing.

In instances where one kind of code finds its way into another kind of code – what I have referred to as "code poisoning" – valuing an author's control over his or her output would at its limit presumably overcome the author's freedom to select from among other authors' work as inputs. Of course, this still does not speak to remedy; one could value strong authorial control without, say, believing in statutory (rather than actual) damages for code poisoning. Indeed, one could imagine a legal framework that provided opportunity to correct poisoned code before damages of any kind accrued.

B. Utilitarian Objectives

1. Innovation.

The proprietary model of software production yields activity encouraged by the availability of exclusive rights, while the existence of public domain and free software products shows that those rights are not always needed to encourage creative output.

What impact would the absence of enforceable exclusive rights have on the innovation currently taking place in the proprietary market? Innova-

---

[23] See Chris Oakes, *Netscape Rolls the Dice*, Wired News (Mar 31, 1998), online at http://www.wired.com/news/technology/0,1282,11358,00.html (visited Dec 16, 2003) (announcing Netscape's plan to release the programming source code for its Communicator software).

[24] See http://www.opensource.org/advocacy/faq.php (visited Dec 16, 2003) ("The Open Source Initiative is a . . . pitch for 'free software' on solid pragmatic grounds rather than ideological tub-thumping.").

tion would decline to the extent that software authors desire money and are relying upon exclusivity to generate income. In the absence of legal support for exclusivity, mercenarily-minded authors might shift more reliance to anti-copying controls and other technical measures to enforce exclusivity. For example, the provision of software could evolve into a service rather than a product: only by "tuning in" to the software author's Internet portal could one run the author's software, and access to this portal could be managed through subscription instead of purchase. A "streaming software" site could then choose to market a lifetime subscription, which would in most respects be functionally equivalent to a purchase. This evolution appears to be taking place in the music industry, where physical instantiations such as cassettes and CDs are giving way to "soft" manifestations such as files located on generic devices or on-demand streaming. From the consumer's point of view, so long as there is ready Internet access, a shift from product to service as a way of technologically proprietizing legally unprotected software could make little difference.

The true value of the exclusive rights to the software publishers may thus lie less in restricting users' behavior, which can be effected through technical means such as copy protection or streaming, and more in restricting the behavior of competing vendors. The separation of source and object code offers a measure of technical protection, ensuring that outside competitors have to go to some expense to reverse engineer targeted software. However, the migration of software engineers—and the source code they have written—from one firm to the next may force companies to fall back on unwieldy non-disclosure agreements or trade secret protection to preserve exclusivity in the absence of copyright. Alternatively, firms could increase the level of compartmentalization to which employees are subjected, giving them less access to company data so that they cannot easily take copies of the firm's goods with them should they go or be lured away. A lessening of available proprietary rights could also stall the nascent sharing of source code by proprietary vendors—a practice still so limited, to be sure, that its loss would be virtually unnoticeable to the market at large. Still, none of these reactions is socially desirable; all would merely contribute to an arms race of protection and circumvention that legal protections in their ideal form (and at the "right" level) pretermit.

What benefits to innovation could loosened proprietary rights bring? One answer lies in the fact that a great deal of software is built literally on predecessors' code. In a well-functioning marketplace of vendors, one can envision individuals and firms who believe they are in a position to improve upon others' work to contract amongst themselves and split the exclusive bounty of the improvement. However, this assumption ignores the fact that such firms are often not readily in negotiating contact with one another, and the absence of available source code may make it hard for outside innova-

tors even to explore the possibility of offering advantageous improvements to existing developers' work.[25]

Putting the still-gelling and much debated issue of software patents aside,[26] if the proprietary model were not legally enforceable, no contact (much less, contract) with the original author would be necessary for an innovator to create a derivative work. This production of derivative works would include the adaptation of formerly proprietary software into public domain software written by downstream authors who do not share the financial incentives or interests that the classic contracting model presumes. With access to source code and an absence of tight legal restriction on its use, nonproprietary authors could help adapt software for audiences whose lack of size or money is insufficient to appeal to vendors driven solely by anticipated profit. Copyright terms of limited duration are one mechanism for splitting the difference between proprietary incentives and later public improvements to formerly proprietary works. However, the effectively infinite copyright term of seventy to ninety-five years for computer software eliminates one source of cheap inputs for both proprietary and nonproprietary authors who wish to make use of others' work that may have already paid for itself (and for which the proprietary model has provided sufficient incentive) several times over.[27]

An author who chooses to place her work in the public domain creates more prospects for innovation than one who requires the incentives of proprietization and who, to the extent allowed, then monopolizes improvements to her work. Such a contribution is "found money" that can be shared nonrivalrously among consumers, and used without further cost or negotiation by other software vendors.[28] But under even the most restrictive plausible copyright regime, so long as the rights conveyed can be repudiated, authors are still free to donate their labors to the overall pool of collaborative labor around free software, and some profit-driven firms have found reasons to want to do so. For example, Netscape made available, with minimal licensing restrictions, the source code of its Communicator 5.0 Web browser software through mozilla.org, a form of at-or-below-cost pricing

---

25 See Mark A. Lemley, *The Economics of Improvement in Intellectual Property Law*, 75 Tex L Rev 989, 1064 (1997) (concluding that the likelihood of original creators and improvers reaching agreement with each other affects what type of default rule is most efficient with regard to their property rights).

26 See generally Bradford L. Smith and Susan O. Mann, *Innovation and Intellectual Property Protection in the Software Industry: An Emerging Role for Patents?*, 71 U Chi L Rev XXX (2004).**[All: Add cite once pagination is finalized. Thx. TFF]**

27 See Jane C. Ginsburg, *How Copyright Got a Bad Name for Itself*, 26 Colum J L & Arts 61, 65 (2002) (discussing the existing struggle between copyright and technology, and expressing skepticism about the value of ever-lengthening copyright terms).

28 This assumes that an author of free software is indeed giving of her time, rather than, say, her employer's. Those working on free software projects as a distraction from day jobs during the day are, perhaps, simply illicitly transferring energy from one model to the other, though employers sufficiently distressed by such behavior could presumably act to stop it.

perhaps designed to staunch the flow of users from Netscape Communicator to Microsoft's Internet Explorer.[29]

Beyond making available the option of placing material into the public domain, do enforceable *copyleft* rights contribute to innovation? The question is, in its essence, a restatement of the initial puzzle: what is the scope of proprietary rights that ought to be created and enforced by the government in the service of innovation? If one believes that innovation primarily happens with financial incentive and favors ensuring such incentive by granting exclusive rights to authors, then an original piece of copylefted software is itself an anomaly, and a requirement that derivative software *not* be proprietary removes incentives to build further upon it.[30] If one additionally believes that free software developers would still write new software even if the copyleft license were not available—forcing a choice between releasing the software into the public domain and claiming traditional proprietary rights—then it would be better not to permit copylefting. A similar question might arise over how much control the legal system should permit donors to assert over the disposition of their gifts, whether given to charities or friends. If donors would still give away money in the absence of an ability to direct its use, one might assume more efficient allocation of the money down the line if the beneficiaries could direct and redirect the funds according to circumstances and preference, weighed against the unhappiness of the donor in not being able to more fully specify the uses for her money.

However, the enforceability of copyleft could assist in "commons creation" in networked software.[31] Copyleft as a mass license attempts to contribute its covered works to a pool of commonly accessible work, and 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. The value of common standards in networked technological endeavors has long been appreciated, and if those standards stood to be proprietized by some future party, current contributors might be tempted to hold back their contributions to the common project.[32]

---

29 See Oakes, *Netscape Rolls the Dice*, Wired News (cited in note 23). The mozilla code appears to have since been completely rewritten by free software developers, and the project is now operated by a non-profit organization. See http://www.mozilla.org (visited Dec 16, 2003).

30 See Chris Sontag, *Is the GPL Good for the Software Industry? No*, Network World 45 (Oct 6, 2003).

31 See generally Nadan, 10 Tex Intel Prop L J at 357–59 (cited in note 1) (discussing how copyleft licenses function to encourage the contributions of a "free labor open source community" that would otherwise be reluctant to share enhancements for free).

32 See, for example, Robert P. Merges, *Who Owns the Charles River Bridge? Intellectual Property and Competition in the Software Industry*, UC Berkeley Public Law & Legal Theory Working Paper No 15, 4–5 (1999), online at http://papers.ssrn.com/abstract=208089 (visited Dec 16, 2003) (discussing the problem of code "lock-out" as stifling innovation in software); Elizabeth G. Lowry, Comment, *Copyright Protection for Computer Languages: Creative Incentive or Technological Threat?*, 39 Emory L J 1293, 1340 (1990) ("With an established [open] standard, developers can concentrate their invest-

The best examples we have of such anxieties in the Internet space, however, have more to do with sudden *retroactive* claims of proprietary rights by formerly cooperating firms.[33] For example, the World Wide Web Consortium has gone to great lengths to ask its member firms to promise not to assert proprietary claims in the standards they contribute to the body. And the networking world has reacted with dismay over the prospect that the International Standards Organization—which previously had asserted copyright in such things as standardized two-letter country code abbreviations solely for the purpose of selling individual documents containing those abbreviations—might now attempt to charge royalties to those firms that have made use of the codes in their software.[34]

Indeed, fundamental Internet protocols are written up in ways designed to ensure continued openness. For example, TCP/IP (by which Internet packets are routed) and SMTP (by which email is sent from one Internet server to another) are described in "request for comments" documents (RFCs) developed in the collaborative model of free software and placed under copyright by the Internet Society on behalf of the unincorporated Internet Engineering Task Force in a way that is designed to prevent anyone from asserting exclusive rights to them. Interestingly, the Internet Society license is consummately ambiguous as to its copyleft status—whether protocols that draw upon those of the Internet Society may themselves be proprietized.[35] At the very least, the Internet Society's assertion of copyright is intended to assure users of the documents that no retroactive claims of more restrictive copyright will be made. It says, in relevant part:

> Copyright (C) The Internet Society (<date>). All Rights Reserved.
>
> This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implmentation [sic] may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works.[36]

---

ments on innovations to a language instead of creation of a new language. Protection of a standard . . . would force competitors to 'reinvent the wheel' in order to develop a competitive product.").

33 See Molly Shaffer Van Houweling, *Cultivating Open Information Platforms: A Land Trust Model*, 1 J Telecommun & High Tech L 309, 314 (2002) (noting that Sun Microsystems won a $20 million settlement after suing Microsoft for developing Sun's Java system into a product that would run only on Microsoft-operated machines).

34 See generally Evan Hansen, *New ISO Fees on the Horizon?*, CNET News.com, online at http://news.com.com/2100-1032_3-5079256.html (visited Dec 16, 2003) (describing an ISO proposal to collect royalties for the use of country codes).

35 See http://www.faqs.org/rfcs/rfc-editor/rfc-copyright-story.html (visited Dec 16, 2003) (indicating that certain types of derivative works are subject to copyleft restrictions, but remaining silent as to derivative works in general).

36 Id.

One cannot tell whether the above license requires that any documents drawing on RFCs enough to be called derivative works—perhaps improvements upon the standards described within—must be released, if at all, in a way that cannot be proprietized. Presumably, though, a company making improvements could simply attempt to rewrite the description at a high enough level of abstraction so as not to constitute a derivative work, and therefore "own" the resulting protocols—derivative to the ideas within the RFC documents, but not derivative to the text of the documents themselves. Even if this is the case, SMTP and TCP/IP may remain safely nonproprietized, unless and until the benefit of adopting a proprietary variant seems unambiguous enough that a critical mass of Internet users, as represented by those writing software that in turn uses the RFC protocols, chooses to adopt it.

On the purely instrumentalist grounds of wanting to foster innovation, there are reasons to expect that different software authors are motivated by different aims. Thus, fostering a system that permits not only public domain works, but also proprietary works and copylefted ones, ensures the maximum range of incentives for those who would write code and share it with others. If a particular innovation cries out for improvement, but the entity best in a position to improve it is confronted with an undesirable initial model—either a proprietary firm wishing to improve upon copylefted software, or a set of free software programmers wanting to improve upon proprietary software—the safety valve of inventing around the restrictions imposed by the model exists. Examples of both can be found. Microsoft rewrote its implementation of Sun's Java programming language from the ground up, leaving Sun Microsystems with only a trademark right, rather than a copyright, in the final product. Symmetrically, Richard Stallman conceived of GNU/Linux as the near-functional-equivalent, but genetically distinct, variant of Unix designed to be free of the original software's copyright claims.[37]

2. Reliability.

It's important to have software that works well. One might think that goal is well accounted for in a simple market model, and perhaps reliability is but a subset of innovation. But there are reasons to want to emphasize it in particular.

The proprietary model boasts such a notion of reliability: presumably consumers will pay exactly for the level of reliability that they want and

---

[37] For a discussion of the complexities involved in evaluating the costs and benefits of inventing around, see Louis Kaplow, *The Patent-Antitrust Intersection: A Reappraisal*, 97 Harv L Rev 1813, 1869–73 (1984) (concluding that inventing around does not contribute to social welfare when patent combinations are permitted, so firms should be forced to compete in order to discourage duplicative research and development and diminish monopoly losses).

that can be cost-effectively delivered to them. Companies like Microsoft have scores of in-house testers, and indeed, in recent months Microsoft has claimed improved reliability[38]—rather than new features—as critical to the success of further iterations of its software.

Eric Raymond's *Cathedral and the Bazaar* essay argues that "peer-reviewed" software—such as collaboratively developed free software in which the source code is viewable, testable, and changeable by all—is more reliable, on the general theory that many disparate eyes can catch more mistakes under more circumstances.[39] Of course, making one's source code available does not guarantee that thousands will flock to view it and fix it. The patching of such software depends on the charitable instincts of volunteer testers, as well as the selfish desires of users of free software to have it function well for their own purposes.

Of course, the development models can overlap somewhat. Proprietary software typically goes through a "beta test," in which consumers are given special pricing (but still no access to source code) in exchange for trying out new software and reporting bugs that are found. Conversely, the most popular free software has variants shepherded by corporate software vendors like Red Hat.[40] These vendors are willing to take responsibility for selecting and distributing improvements to users, and provide general support and consultation to those users on a cash-and-carry basis.

An attempt to generalize about the innate superiority of the free development model over the proprietary one along the axis of reliability is difficult, and the debate in large part turns on which empirical examples are chosen to support each side. Still, free and proprietary software can compete alongside one another in a market; market participants wanting to run a Web server can decide between installing the free Apache software at one price or installing Microsoft software at another price, and can factor into the purchasing decision estimates of reliability or ease of addressing later problems.

But there is a more important way in which reliability is at issue, one in which individual market decisions may prove insufficient. Twenty-first century software is in many cases not installed and used in isolation. Rather, the machines running the software are connected to and communicating across the global Internet. To the extent those machines run identical software, a single flaw in that software can be exploited across the network, and the total cost of software unreliability must be taken into account. That total may not be measured only by the sum of harms to each compromised user. For example, if one is shopping online among three vendors of books

---

38 See http://www.microsoft.com/mscorp/innovation/twc/ (visited Dec 16, 2003) (describing Microsoft's launch of Trustworthy Computing, an initiative aimed at improving "security, privacy, reliability, and business integrity").

39 See Raymond, 3 First Monday (cited in note 17).

40 See http://www.redhat.com/ (visited Dec 16, 2003).

and all three vendors' sites disappear at the same moment because each is running the same flawed software, succumbing to the same maliciously exploited flaw, the inconvenience is far worse than if each vendor were to experience the same problems but at different times, for different reasons, because each was running different software.

Further, a computer worm may instruct each infected host to generate new network traffic, perhaps seeking new computers to infect. That traffic slows down the network for everyone—accruing costs even to those not infected or who, because they are running completely different software, cannot be infected. Homogeneity in deployed software scores well for interoperability among machines, but it places everyone's eggs in one basket. A diversity of computer platforms running a diversity of software distributes those eggs—perhaps resulting in the same number of potential infections but avoiding catastrophic simultaneous infection.

The notion that free software could come to exist even when the market does not otherwise call for it in strict dollar calculus—as in Richard Stallman's GNU/Linux ideological endeavor to *recreate* Unix functionality without using any Unix code—is a windfall insurance policy beyond whatever competitive benefits can come when multiple market-driven vendors respond to financial incentives to write competing code.

3. Use and dissemination.

Were there not a perceived need to create monetary incentives for its production through monopoly control of code, the nonrivalrous nature of software would make for an abundance of existing code that could easily satisfy every demand if copying were freely permitted. The legal rights of the proprietary model of production are geared to create those incentives, but they come with a well-documented cost. Tempered only to some extent by crude price discrimination, the monopoly holders of rights in proprietary software will end up making available fewer copies of their software than there are users who want them—users who would pay a price for them that still beats the vendor's marginal cost. This has been used as one basis of claiming that leaky enforcement of copyright can actually assist the proprietary vendor while enhancing social welfare, since poorer consumers can come together to pool their money to purchase and then—perhaps illegally but still beneficially—share a single copy of a proprietary work.[41]

If someone truly cannot afford the software, a vendor might be indifferent to that person's obtaining and using it, since it cannot be realistically counted as a lost sale. Indeed, there may be reasons why it is helpful to get someone who might be a future customer to use one's software at a time

---

[41] See Yannis Bakos, Erik Brynjolfsson, and Douglas Lichtman, *Shared Information Goods*, 42 J L & Econ 117, 123–124 (1999) (stating that team valuations make consumer purchasing more predictable, at times increasing seller profit).

when he or she otherwise would not pay for it. This could be for the purposes of acclimatization in anticipation of future paying use, such as in the cases of free access to services offered to law students by both Westlaw and Lexis. It could also be used, when network effects are thought to be present, as a way of creating systemic momentum towards one's increased market share of products by non-paying and paying customers alike. The increasing willingness on Microsoft's part to cut its prices in developing countries may be an example of this, as a way of forestalling competing software—especially GNU/Linux—from taking root there. And, thanks to network effects between platform and application, there can be reasons to want to see wide adoption of certain components of one's stable of software. Microsoft's and Netscape's Internet browsers remain examples of a giveaway in service of such a rationale.[42] In Microsoft's case it was likely to defeat a threat to its operating system by the Java computing platform bundled with the competing Netscape browser; in Netscape's case it was largely to make sales of Netscape Web servers seem more attractive, given the number of Netscape browsers that might be thought to more seamlessly connect to them.

But if equal access is sought, the free and public domain models by definition guarantee such opportunity. If one were more concerned about an even distribution of technology's fruits—rather than in providing monetary incentives for the creation of those fruits to begin with—then any scheme that eschews assertion of exclusive rights over its products would seem preferable along this dimension. Copylefting one's software may provide better access than simply releasing it into the public domain, since it insists that any works based upon the copylefted software themselves be made available without exclusive restriction.

C. Accounting for Differential Systemic Legal Vulnerabilities

Should systemic differences in levels of "legal aggressiveness" between the competing models be adjusted for? Major vendors of proprietary software have a stake, with their counterparts in other proprietary creative fields, in defending the overall system of rights that produces the greatest revenue. They hire lobbyists, donate to congressional campaigns, and ask federal trade representatives to vindicate their structural rights overseas. When the World Intellectual Property Organization announced the prospect of a meeting devoted to studying the place of nonproprietary production models within the spectrum of intellectual property,[43] the Business Software

---

[42] See, for example, Jonathan Zittrain, *The Un-Microsoft Un-Remedy: Law Can Prevent the Problem that It Can't Patch Later*, 31 Conn L Rev 1361, 1365–66 (1999), for strategic reasons that Microsoft and others might choose to give away a browser for free.

[43] The author co-signed a letter urging WIPO to hold such an exploratory meeting. See July 7, 2003 Letter from Sixty-Eight Scientists and Economists to Kamil Idris, Director General of the World Intellectual Property Organization, online at http://www.cptech.org/ip/wipo/kamil-idris-7july2003.pdf

Alliance objected strenuously, and the meeting was canceled.[44] Of course, free software advocates are known to function in political circles—there are trade associations favorable to the free software idea[45] and numerous academics making the case for models of nonproprietary production.[46]

Apart from a more general political debate between the competing spheres in which proprietary software's profits may have an advantage over free software's prophets, there is a discernable "litigation differential." Prior to litigation derivative to the *SCO v IBM* case, there were only two recorded instances of litigation initiated by holders of copyleft licenses claiming proprietization of code in violation of the license.[47] This dearth of litigation may be traced to the lack of violations, but it might also be plausibly attributed to the fact that proprietary software does not make available its source code—making it difficult to detect if such software contains an infringement of copylefted code. Free software, on the other hand, is much more vulnerable to claims of infringement by proprietary code authors, since the source code to free software is, by definition, available for examination by would-be plaintiffs. That availability also makes the costs of stealing copylefted software typically lower than the costs of stealing proprietary code, since free software's source code is there for the taking.

Further, the collaborative nature of free software development makes it harder to determine where various contributions are coming from, and whether they belong to those who purport to donate them. Indeed, in the example of an employee of a software company charitably moonlighting for a free software project, the employee's work may not even be the employee's to give. A barely-read but still facially enforceable employment agreement may commit all software written by the employee to the employer's possession, which would set the stage for an infringement claim against those within the free software project making use of the employee's contributions. Of course, the collaborative nature of free software development may simply mean that more violations can take place, in which case

---

(visited Dec 16, 2003) (requesting that WIPO host a meeting on open and collaborative development).

44 See Jonathan Krim, *The Quiet War Over Open-Source*, Wash Post E1 (Aug 21, 2003).

45 See, for example, The Consumer Project on Technology, http://www.cptech.org (visited Dec 16, 2003) (a non-profit advocacy group started by Ralph Nader in 1995 focusing on a variety of technology issues affecting consumers, including intellectual property rights and competition policy).

46 See, for example, Yochai Benkler, *Coase's Penguin, or, Linux and the Nature of the Firm*, 112 Yale L J 369, 381 (2002) (stating that peer production is not always the best model, but it does have certain systematic advantages that allow it to succeed); Larry Lessig, *The Future of Ideas: The Fate of the Commons in a Connected World* 14–16 (Random House 2001) (arguing that control stifles creativity, and that just because control is possible does not mean it is justified).

47 See *FAQ on MySQL vs. NuSphere Dispute* (Jul 13, 2001), online at http://www.mysql.com/news/article-75.html (visited Dec 16, 2003), for the plaintiff's view of one case. The case settled on undisclosed terms. See also John Markoff, *Copyright Lawsuit Is Turnabout for SCO*, NY Times C2 (Oct 13, 2003) (describing a lawsuit (the settlement of which is under seal) brought by a corporate maker of copylefted software against a sister company to SCO).

increased legal vulnerability to claims of infringement would be a natural consequence rather than an indication of an unlevel playing field.

Finally, the costs of litigation are beyond the reach of many free software developers—who may be donating their time, but do not expect to have to spend or receive hard cash as a result of their labors. The idea of providing legal shelters for noncommercial or nonprofit entities—in the form of tax-exempt status, charitable immunity from tort, or as a factor in a fair use test excusing copyright infringement[48]—is not new, and may reflect some desire for such enterprises to be able to devote their energies to their eleemosynary purposes. Of course, such breaks are often controversial,[49] but a systemic vulnerability of free software to theft by proprietary companies, coupled with a comparatively higher exposure of free software authors, publishers, and even mere users to accusations of theft by proprietary software companies, suggests a playing field that is not level.

CONCLUSION

SCO v IBM will be the first case to test the legal viability of the free software development model, and its concomitant practice of making software source code routinely available to the general public. It most likely will not be the last. In future work, I will explore the possibility that, under the normative principles described here, copyright law should be construed in a way that does not permit a poisoned pea of unauthorized code under the mattress of a massive software project to effectively compromise the entire work. I will consider the possibility that copyright's statute of limitations might be applied to require those claiming copyright infringement to bring such claims within a three-year (or shorter) window stemming from the targeted software's initial public release of source code, encouraging creators within both models to release their source code, and providing helpful legal stability to those wishing to work within a collaborative software development environment.

Both free and proprietary software production have a storied and venerable history dating back to the first moments of public computing. Ensuring a demilitarized zone between them serves most of the interests we would care to advance. It shows caution in the face of uncertainty about the long-term benefits of either model against the other and demonstrates an appreciation of the subtlety of the conflict between them.

---

[48] See 17 USC § 107(1) (2000) (providing that "the purpose and character of the use, including whether such use is of a commercial nature or is for nonprofit educational purposes" should be one of the factors considered in determining whether a particular use of a copyrighted work is a "fair use").

[49] See, for example, Tanya D. Marsh, *A Dubious Distinction: Rethinking Tax Treatment of Private Foundations and Public Charities*, 22 Va Tax Rev 137, 148–50 (2002) (discussing the history of controversy surrounding tax treatment of private foundations and Congress's attempts to thwart abusive tax shelters).