"INFECTIOUS" OPEN SOURCE SOFTWARE: SPREADING INCENTIVES OR PROMOTING RESISTANCE?

Greg R. Vetter[*]

ABSTRACT

Some free or open source software infects other software with its licensing terms. Popularly, this is called a viral license, but the software is not a computer virus. Free or open source software is a copyright-based licensing system. It typically allows modification and distribution on conditions such as source code availability, royalty free use and other requirements. Some licenses require distribution of modifications under the same terms. A license is infectious when it has a strong scope for the modifications provision. The scope arises from a broad conception of software derivative works. A strong infectious ambit would apply itself to modified software, and to software intermixed or coupled with non-open-source software. Popular open source software, including the

[*] Assistant Professor of Law, University of Houston Law Center (UHLC); Co-Director, Institute for Intellectual Property and Information Law (IPIL) at UHLC; biography and additional background available at: www.law.uh.edu/faculty/gvetter. Relevant to this Article is that my background includes a Master's degree in Computer Science and full-time work experience in the business-to-business software industry from 1987 to 1996. Research for this Article was supported by summer research grants from the University of Houston Law Foundation and a grant from the University of Houston's New Faculty Research Program. I also thank UHLC's IPIL Institute and its sponsors for support of my endeavors at UHLC. My thanks to UHLC students Jason Williams, Cuong Lam Nguyen, Stacey R. Vause and Nivine Zakhari for research assistance. My special thanks to UHLC student Kristin Brown for comments, suggestions and research assistance on this Article. I also benefited from input and discussions with Joe Dryer, a UHLC alumnus whose Houston-based company is involved in open source software applications. In addition, I am thankful for exceptionally capable support provided by the staff of the University of Houston Law Center's John M. O'Quinn Law Library, including Peter Egler and Nicole Evans, and by UHLC's Legal Information Technology Department. My special thanks to Craig Joyce, Ray Nimmer, Heather Meeker, and Ron Chichester for helpful discussion and comments. Finally, I thank Peter Yu, Director of the Intellectual Property and Communications Law Program at Michigan State University College of Law. I presented elements of this Article at that program's Conference on Intellectual Property, Sustainable Development, and Endangered Species: Understanding the Dynamics of the Information Ecosystems. My thanks to audience members and co-panelists for comments and feedback.

GNU/Linux operating system, uses a license with this feature. This Article assesses the efficacy of broad infectious license terms to determine their incentive effects for open source and proprietary software. The analysis doubts beneficial effects. Rather, on balance, such terms may produce incentives detrimental to interoperability and coexistence between open and proprietary code. As a result, open source licensing should precisely define infectious terms in order to support open source development without countervailing effects and misaligned incentives.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

I. INTRODUCTION

> Copyright offers both incentives and deterrents to derivative uses, protecting derivative works as well as prohibiting their unauthorized production.
>
> Paul Goldstein,
> *Derivative Rights and Derivative Works in Copyright*[1]

> The GPL requires that works "derived from" a work licensed under the GPL also be licensed under the GPL. Unfortunately what counts as a derived work can be a bit vague. As soon as you try to draw the line at derived works, the problem immediately becomes one of where do you draw the line?
>
> Linus Torvalds,
> *The Linux Edge*[2]

Too often, the open source software debate has an all-or-nothing flavor.[3] One might hear: open source software is better because the production process taps debugging efficiencies and harvests volunteer effort on an impressive scale.[4] Or hear: proprietary software is better because it

[1] Paul Goldstein, *Derivative Rights and Derivative Works in Copyright*, 30 J. COPYRIGHT SOC'Y U.S. 209, 239 (1983).

[2] Linus Torvalds, *The Linux Edge*, *in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION 108-09 (Chris DiBona et al. eds., 1999) [hereinafter Torvalds, *The Linux Edge*].

[3] *See* Peter Wayner, *Whose Intellectual Property Is It, Anyway? The Open* Source *War*, N.Y. TIMES, Aug. 24, 2000, at G8 (describing that "the war of Open Source . . . is being fought in conference rooms, law offices, hacker redoubts and university dormitory rooms and in the hearts of millions of people surfing the Web"); Eben Moglen, Freeing the Mind: Free Software and the Death of Proprietary Culture, Address Before the University of Maine Law School's Fourth Annual Technology and Law Conference (June 29, 2003) (describing the "fundamental choice" between free software and a "dead [proprietary] system of inefficient distribution"), *at* http://emoglen.law.columbia.edu/publications/maine-speech.pdf; *see also* David McGowan, *SCO What? Rhetoric, Law and the Future of F/OSS Production* 10-16 (Univ. of Minn. Law School, Research Paper No. 04-9, June 12, 2004) [hereinafter McGowan, *SCO What?*] (describing the rhetorical strategies used by both sides of the debate and their accompanying exaggerated claims), *available at* http://www.law.umn.edu/uploads/images/830/McGowanD-SCOssrn.pdf.

[4] *See, e.g.*, Yochai Benkler, *Coase's Penguin, or, Linux and* The Nature of the Firm, 112 YALE L.J. 369, 376-77, 415-23 (2002) (noting that one advantage of open source software and peer production in general is that "it allows larger groups of individuals to scour larger groups of resources in search of materials, projects, collaborations, and combinations than is possible for firms or individuals who function in markets"); Eric S. Raymond, *The Cathedral and the Bazaar* (2000) (coining the phrase "given enough eyeballs, all bugs are shallow," also

promotes traditional software business and investment models, and software has seen rapid growth.[5]

The debate gives insufficient credence to coexistence between the two types of software.[6] This inhibits progress toward an optimal presence for both types in the greater software ecosystem, both domestically and on the international scene. The growth trajectory of open source software suggests that it is a long-term player in that ecosystem, provided there is market confidence in the mode of legal protection underlying open source software.

Currently, open source software uses a copyright-based, generally applicable license to grant permissions for others to use, modify and distribute the software. In return, licensees must comply with conditions that keep the software "open source."[7] A variety of open source software

known as "Linus' Law"), *at* http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar (last modified Aug. 2, 2002); *see also* David McGowan, *Legal Implications of Open-Source Software*, 2001 U. ILL. L. REV. 241, 268, 274 (2001) [hereinafter McGowan, *Legal Implications*] (noting the volunteerism underlying open source software development).

[5] David S. Evans, *Politics and Programming: Government Preferences for Promoting Open Source Software, in* GOVERNMENT POLICY TOWARD OPEN SOURCE SOFTWARE 34, 35-37 (Robert W. Hahn ed., 2002) (discussing the high rates of productivity, innovation, and competition that the proprietary development method has brought to the software industry), *available at* http://aei-brookings.org/admin/pdffiles/phpJ6.pdf (last visited Jan. 8, 2004).

[6] Bradford L. Smith, *The Future of Software: Enabling the Marketplace to Decide, in* GOVERNMENT POLICY TOWARD OPEN SOURCE SOFTWARE, SUPRA NOTE 5, AT 69-70 (discussing the over two decade long debate about the merits of open source software, but arguing that "*both* open source and commercial software are integral parts of the broader software ecosystem") (emphasis in original). *But see* Jonathan Zittrain, *Normative Principles for Evaluating Free and Proprietary Software*, 71 U. CHI. L. REV. 265, 266, 282 (2004) (positing on the one hand that the "legal forms of proprietary and free software production cannot coexist within a given piece of code" but observing on the other that "free and proprietary software can compete alongside one another in a market").

For a general discussion of open source and Linux deployment in corporate computing environments, see MARTIN FINK, THE BUSINESS AND ECONOMICS OF LINUX AND OPEN SOURCE 119-33 (2003).

[7] *See* Zittrain, *supra* note 6, at 269 (describing conditions that keep software "open source" as a "form of legal jujitsu" to protect free software from proprietary software developers).

Throughout this Article I deal with open source software as follows: characterizing it as a licensing system that promulgates generally applicable, conditional permissions. The other possible characterization is that these licenses will in some instances create contracts among developers, distributors and users. While the contract analysis is important to open source software, this Article does not delve into the many doctrinal questions of contract law that the various open source software licenses raise. *See* McGowan, *Legal Implications*, *supra* note 4, licenses exist. The General Public License, or GPL,[8] is the most prominent license. It is the first of its kind. The GPL contemplates a buffer zone. If proprietary software falls into this zone when intermixed with the open source software, it must henceforth be distributed as open source software. This is sometimes called an "infectious" or "viral" license,[9] but this does not mean that the software itself is a virus or is malicious.

This Article examines infectious license terms of expansive scope and the incentives such terms create for open source and proprietary software. Some estimate that infectious terms promote open source software growth by

at 289-302 (describing and analyzing a number of the potential doctrinal questions of contract law raised by the GPL); Greg R. Vetter, *The Collaborative Integrity of Open Source Software*, 2004 UTAH L. REV. 563, 644-47 (cataloging potential issues from treating open source licenses as agreements and noting that the questions raised are sometimes similar to enforceability questions for shrinkwrap licenses).

Often one cannot determine from the face of the license whether its deployment will create a contract. More facts might be needed, such as whether there was any explicit or implicit assent to the terms, or whether the open source software provided full notice of the license's terms to downstream licensees. Thus, I treat the license for its core purpose, as a defense to a copyright infringement action. *See* David McGowan, *Legal Aspects of Free and Open Source Software* 4-7 [hereinafter McGowan, *Legal Aspects*] (describing that the primary enforcement mechanism for open source licenses is the copyright infringement power), *available at* http://www.law.umn.edu/uploads/images/253/McGowanD-OpenSource.rtf (Oct. 5, 2004). I mostly put aside questions such as whether, if treated as a contract, a particular open source license would give the licensor rights against those who take the software, or whether lack of privity eliminates any recourse an original licensor might have under contract against sub-licensees down a chain of distribution.

On the other hand, in either a contract sense or a permission sense, for infectious terms, ultimately a court will be interpreting words in a written instrument. Even if such interpretation is not under strict contract law, courts may borrow contract interpretation principles. For broadly-phrased infectious terms, then, a key source of the uncertainty will spring from the same source in both perspectives: the license's broad language expressing the infectious terms.

[8] GNU General Public License Version 2 (June 1991) [hereinafter GPL], *available at* http://www.gnu.org/licenses/gpl.html.

[9] This Article's title uses the word "infectious" to intonate the license characteristic that I study. In my vocabulary, this replaces the common phrase "viral license" for several reasons. First, the Article focuses on the extension of license conditions when software is intermixed and coupled. This is more consistent with my focus than the carrying of such terms through a chain of software distribution. In the medical context, a virus is a carrier. Second, I find "infectious" less pejorative than "viral." I find no positive connotations for "viral" or "virus." While this Article argues against broad infectious license terms, it and my previous open source software article reflect my positive view of the movement. There are at least some positive connotations for "infectious": the positive energy behind the open source movement is infectious, like laughter, or like the feeling of a job well done.

supporting community development norms and preventing proprietary poaching of the software, or converting proprietary software to open source. Even if some of these viably benefit open source, my analysis is skeptical that infectious terms, on balance, have beneficial effects. They adversely increase legal uncertainty for open source licenses and produce incentives detrimental to interoperability, compatibility, and coexistence between open and closed source software.[10] As a result, open source licensing should precisely define infectious terms in order to support open source development without countervailing effects and misaligned incentives.

In particular, the analysis challenges the third purported benefit of infectious terms: converting proprietary software to open source. Broad infectious terms are unlikely to cause such conversion. Some proprietary software may adopt the open source development model for its particular benefits and attributes. Such vendors, however, will do this on their own timeline and inclination. They are unlikely to succumb to surprise infection. The GPL, with its infectious terms, is the most widely adopted open source software license. However, many seek license terms that are not infectious, or they seek terms that more clearly delineate the scope, and corresponding risk, of the infectious terms. This is shown by other popular licenses without infectious terms. It is underscored by the steady push of open source software into corporate and enterprise computing. These risk-minimizing groups prefer greater certainty in their duties and obligations. While infectious terms may be necessary to some degree to support the other conditions that keep software open source, the debate shows that broad, infectious terms of expansive scope have a polarizing influence.

An example of this all-or-nothing debate is illustrated by the popular press. The *Wall Street Journal* ran an article with a cartoon showing a small portly penguin swinging a slingshot, taking aim at a giant.[11] The penguin's name is not David, but Tux, the mascot of the open source operating system

[10] *See generally* JONATHAN BAND & MASANOBU KATOH, INTERFACES ON TRIAL: INTELLECTUAL PROPERTY AND INTEROPERABILITY IN THE GLOBAL SOFTWARE INDUSTRY xxi-xxii (1995) (describing the history of legal developments as they bear on interoperability among software applications and between software with hardware).

[11] Robert A. Guth, *Free to Choose*, WALL ST. J., May 19, 2003, at R6.

known as GNU/Linux.[12] In the cartoon, however, Goliath was labeled. His belt bears the moniker Microsoft and his shield carries the Microsoft Windows logo.[13] Goliath stands, knees buckling, as Tux prepares to sling another stone.[14] The cartoon vividly illustrates the popular perception of a marketplace battle between the flagship open source software product and the world's largest software company.[15] There is strenuous competition between these two in niche markets. However, proprietary software, much of it internally custom-developed, still dominates the software ecosystem.[16]

[12] *Id.* Here, I clarify my use of a few important terms. The GNU/Linux operating system is popularly referred to as Linux. The operating system, however, is not a single large software work, but is rather an aggregation of many software components. The central component is the kernel, which is properly called Linux. Distributions of a Linux-kernel-based operating system include other critical components. Most distributions include a set of essential software tools from the GNU project, a separate open source software effort. Thus, the most accurate attribution uses the name GNU/Linux for such a distribution.

Another term that varies both in popular use, and among the open source community, is whether one should call the software "free" software, "open source" software, or perhaps "free/libre and open source" software, sometimes referred to as "FLOSS," "FOSS" or "f/oss." The "free" label is apt for licensing systems that disallow charging royalties for use of the software, as most open source software licenses do. "Libre" is the preferred international label popularized by the European Commission, which emphasizes the alternate usage of "free" as in liberated, for which "libre" is the French and Spanish equivalent. *See* Int'l Inst. of Infonomics, Free/Libre and Open Source Software: Survey and Study, *at* http://www.infonomics.nl/FLOSS/? (last visited Sept. 20, 2004). To shorten and simplify the label, I will simply use "open source software" to include "free/libre and open source software."

[13] Guth, *supra* note 11, at R6.

[14] *Id.*

[15] Linux's success is most vividly illustrated in the submarket for "server" computers, which underlie much of the Internet. In the server market, Linux shipments are estimated to climb to 15.9% during 2003 from 13.3% in 2002. *Id.* Microsoft's estimated shipments, however, were essentially flat from 2002 to 2003, at approximately 60.4%. *Id*. In the overall operating system market, which by sheer numbers is dominated by "desktop" computers running Microsoft's Windows operating system, Linux share is at about two percent, but growing. Margret Johnston, *IDC: Microsoft Tightens Vise on OS market*, COMPUTERWORLD, Feb. 28, 2001, *available at* http://www.computerworld.com/governmenttopics/government/legalissues/story/0,10801,58278,00.html; *see also* United States v. Microsoft Corp., 84 F. Supp. 2d 9, 22-24 (D.D.C. 1999) (analyzing open source software as a potential market entrant and noting that "[a]lthough Linux has between ten and fifteen million users, the majority of them use the operating system to run servers, not PCs"), *aff'd in part, rev'd in part*, 253 F.3d 34 (D.C. Cir. 2001).

[16] James Bessen, *What Good is Free Software?*, *in* GOVERNMENT POLICY TOWARD OPEN SOURCE SOFTWARE, *supra* note 5, at 12, 17-24 (discussing some recent successes of

As the flagship open source software application, Tux and GNU/Linux hope to lead the way for greater use of open source generally, and of GNU/Linux specifically.

A complex case, however, threatens to ground Tux and GNU/Linux in these efforts. A small software company, SCO, seeks recourse from IBM, claiming that IBM injected SCO's proprietary software into the GNU/Linux operating system without permission and thereby breached contracts between the two.[17] IBM distributes GNU/Linux. Along with thousands of other programmers and organizations around the world, IBM contributes open source code to the operating system. If the claims are true, SCO may have a copyright infringement action against GNU/Linux end users who will number in the tens, or even hundreds, of thousands.[18] Thus, the situation

open source software, but noting that "the majority of software produced" is proprietary software that is "either self-developed or custom").

[17] *See* Steve Lohr, *No Concession from I.B.M. in Linux Fight*, N.Y. TIMES, June 14, 2003, at C1 (describing the suit and noting concern caused by the suit among corporate technology buyers), *available at* http://www.nytimes.com; *SCO Files Suit Against IBM*, *at* http://www.sco.com/ibmlawsuit (last visited Oct. 5, 2003) (giving plaintiff's web site which describes the suit and providing links to documents filed by SCO); *see also* McGowan, *SCO What?*, *supra* note 3, at 17-22 (describing and analyzing certain issues raised by the *SCO* case).

[18] Indeed, SCO has followed its allegations against IBM to this conclusion and has announced plans to seek licensing fees from certain institutional Linux users. *See* David Bank, *SCO Announces Plans to Seek Licensing Fees from Linux Users*, WALL ST. J., July 22, 2003, at B5 (noting that SCO has retained David Boies, an attorney "who played a major role in the government's antitrust case against Microsoft"), *available at* 2003 WL-WSJ 3974680. The move against users is related to the suit against IBM. SCO has a copyright infringement case against users of Linux if its allegations against IBM are true – that IBM (or others, for that matter) in fact incorporated SCO's code into Linux without permission.

The following two examples illustrate how SCO followed through on its announcement to seek recourse from Linux users. During the first week of March, 2004, SCO sued AutoZone and DaimlerChrysler for using Linux. *See* Complaint at 1, SCO Group, Inc. v. Autozone, Inc. (D. Nev. Mar. 3, 2004) (No. CV-S-04-0237-DWH-LRL) [hereinafter AutoZone Complaint], *available at* http://sco.tuxrocks.com/Docs/AZ/AZ-0.pdf (last visited Oct. 17, 2004); Complaint at 1, SCO Group, Inc. v. DaimlerChrysler Corp. (Mich. Cir. Ct. Mar. 3, 2004) (No. 04-056587-CK) [hereinafter DaimlerChrysler Complaint], *available at* http://www.groklaw.net/pdf/sco-dcx.pdf (last visited Oct. 17, 2004); *see also* David Bank, *SCO Broadens Its Attack on Linux*, WALL ST. J., Mar. 4, 2004, *available at* 2004 WL-WSJ 56921950.

Although both actions spring from the use of Linux, they are different. SCO's suit against AutoZone is in federal district court, based on copyright infringement. AutoZone Complaint, at 6-7, ¶¶ 20-22. As of October, 2004, the action against AutoZone is stayed threatens customer confidence in the copyright-based licensing system, underpinning not only GNU/Linux, but also many other open source software products.[19]

pending the outcome of the IBM, Novell, and Red Hat cases, although the court has allowed SCO and AutoZone to take "limited expedited discovery related to the issue of preliminary injunctive relief." SCO Group, Inc. v. Autozone, Inc., No. CV-S-04-0237-RCJ-LRL, at 1-2 (D. Nev. Aug. 6, 2004) (order granting stay motion), *available at* http://www.groklaw.net/pdf/AZ-35.pdf (last visited Oct. 17, 2004); *see also* Bob Mims, *Judge Dismisses Utah Software Firm's Suit Against DaimlerChrysler*, SALT LAKE TRIB., July 22, 2004, *available at* 2004 WL 59302116. The stay order also required the parties to "submit a letter to the [c]ourt every [ninety] days as to the status" of SCO's cases against IBM, Novell, and Red Hat. *Autozone*, No. CV-S-04-0237-RCJ-LRL, at 1. AutoZone then filed an emergency motion for stay of discovery, which was denied in September, 2004. However, the judge indicated that if Novell wins in SCO's case against it, then the AutoZone case would be over. *Autozone*, No. CV-S-04-0237-RCJ-LRL (D. Nev. Sept. 23, 2004) (order denying AutoZone's emergency motion for stay of discovery). For an unofficial transcript of the hearing on Autozone's emergency motion for a stay, see http://www.groklaw.net/article.php?story=20040910123928788.

The suit against DaimlerChrysler is in Michigan state court, based on a license agreement that SCO has with DaimlerChrysler. DaimlerChrysler Complaint, at 5, ¶¶ 19-20. SCO alleges that DaimlerChrysler is in breach of the agreement for failing to provide a certification that it is not in violation of the agreement's provisions regarding the use of Linux. *Id.* at 7, ¶¶ 27-28. DaimlerChrysler filed a motion for summary disposition on the same day it answered the complaint. Motion for Summary Disposition, SCO Group, Inc. v. DaimlerChrysler Corp. (Mich. Cir. Ct. Apr. 15, 2004) (No. 04-056587-CKB), *available at* http://www.sco.tuxrocks.com/Docs/DC/DC-2004-04-15-A.pdf (last visited Oct. 17, 2004); Answer and Affirmative Defenses, *SCO Group, Inc.* (No. 04-056587-CKB), *available at* http://www.groklaw.net/pdf/DCAns.pdf (last visited Oct. 17, 2004). All of SCO's claims against the auto manufacturer were eventually dismissed, except for the claim relating to whether DaimlerChrysler was late in responding to SCO's demand to provide certification that it had not violated the terms of the agreement. Mims, *supra*; Scott Morrison, *Court Blow for SCO's Linux Campaign Software*, FIN. TIMES, July 22, 2004, *available at* 2004 WL 87882502.

[19] A definition of the term "open source software" is in order here. The term's meaning arises in part from a technological aspect of software: it can have a source code form, or an object code form. Putting aside innumerable technical distinctions, humans can read the source code form and use it to write software. The source code is then processed or compiled into object code form, which a computer can read and execute. Possession of the software's source code allows one to examine the internal operation of the program to see how it was designed and written. If a person has only the object code, doing this is much more difficult. *See* Vetter, *supra* note 7, at 578-82 (relating a model of computing to the source and object code forms of software and discussing the resulting implications).

Traditionally, most software provided users only the object code form of the computer program. Thus, the term "open source software" signals that the source code form is also provided or made available. In this sense, the source is "open." There are other nuances to the term's meaning, but at bottom it means that source code is available. This is important

Even if SCO’s claims are eventually shown to be baseless, or that the users have viable defenses, the suit endangers Tux’s momentum and growth. Some may see that the only bright side to the suit is that it may jolt the open source software community to reexamine licensing and other practices to preventively counter such problems in the future.[20] However, the suit also puts the GPL’s infectious terms front and center in the open source debate. One IBM countermove is its allegation that SCO has itself distributed GNU/Linux under the GPL, and thus SCO’s claim to proprietary rights in the code violates the GPL.[21] In other words, under this theory, any proprietary

because such availability enables distributed, collaborative computer program development among far-flung programmers with lesser centralized control than traditionally found in software development projects. *See id.* at 567-68 (noting that the availability of source code over the internet allows open source programmers to share source code royalty-free and collaborate in self-organized groups to develop software); *see also* McGowan, *Legal Implications*, *supra* note 4, at 253. This mode of development is a hallmark of the open source movement, and a progenitor of its success. *See generally* Benkler, *supra* note 4, 434-36 (postulating a formal model describing collaborative peer production for information outputs such as open source software).

[20] *See* John C. Dvorak, *Killing Linux: Linux and the Whole Open-Source Movement are in Peril*, PC MAG., June 1, 2003, *available at* 2003 WL 5729467 (arguing that the open source community is neither taking the suit seriously enough, nor addressing the underlying risk of open source programming – that a contributor who, unbeknownst to the rest of the group, injects protected code into a project).

In the summer of 2004, the Linux operating system kernel group, under the direction of Linus Torvalds, implemented a new practice: requiring programmers who submit software code to the project to include a “Developer’s Certificate of Origin” field in order to track submissions and attribute them to the submitting programmer. *See* Larry Greenemeier, *Linux Process Change Raises Questions*, INFO. WK., May 31, 2004, *available at* 2004 WL 61411477.

[21] IBM’s sixth counterclaim alleges that SCO breached the GNU General Public License, in part by “seeking to impose additional restrictions on the recipients of programs licensed under the GPL, including IBM contributions, distributed by SCO.” Counterclaim at 33-34, SCO Group, Inc. v. Int’l Bus. Machs. Corp. (D. Utah 2004) (No. 2:03CV-0294 DAK) [hereinafter IBM Counterclaim], *available at* http://www.sco.com/ibmlawsuit/20040329_ibm_2nd_amended_counterclaim.pdf (last visited Oct. 17, 2004); s*ee* Stephen Shankland, *Judge Orders SCO to Show Linux Infringement, at* http://news.com.com/2102-7344_3-5114689.html (last modified Dec. 5, 2003) (discussing a court order requiring SCO to produce information in response to an interrogatory relevant to IBM’s argument “that SCO’s distribution of a Linux product means it has agreed to the GPL’s terms and therefore given permission to use that particular Unix technology in Linux”); *see also* Eben Moglen, *SCO: Without Fear and Without Research* 3-4 (2003), *at* http://www.osdl.org/docs/osdl_eben_moglen_second_statement.pdf (last visited Oct. 17, 2004) (arguing that IBM’s defense of asserting the GPL against SCO is likely to be effective).

SCO code would be converted to open source software under the GPL's infectious terms.[22]

If the SCO suit suggests reexamining licensing and other practices for open source software, this reexamination should include the difficult questions raised by this suit about optimal methods, from a licensing perspective,[23] for open source software to coexist with proprietary software.[24]

[22] IBM alleges that SCO distributed Linux products under the GPL, and "[b]y so doing, SCO accepted the terms of the GPL . . . both with respect to source code made available by IBM under the GPL and with respect to SCO's own Linux distributions." IBM Counterclaim, *supra* note 21, at 33.

When someone intentionally publishes software for distribution under the GPL, the code will be open source software. *But see* McGowan, *Legal Aspects*, *supra* note 7, at 13-14 (noting that the GPL does not specify a term for the license permission, which raises the issue of the licensor's power to terminate the license: in some jurisdictions, without a specified term, the license is terminable at will by the licensor). In arguing that the GPL would not apply to convert its code (alleged to be in GNU/Linux) to open source software, SCO relies on its assertion that IBM put the code into GNU/Linux without SCO's authorization. SCO is in effect saying that it should not matter if later it took GNU/Linux and redistributed it because any SCO code in the operating system was put there by others. It is possible that a GNU/Linux distribution could have passed through SCO without it ever knowing its code was (if it was) in the distribution; GNU/Linux has millions of lines of source code. Open source software distributors often do not assess and evaluate every line of code merely to redistribute the product.

[23] *See* Robert W. Gomulkiewicz, *De-Bugging Open Source Software Licensing*, 64 U. PITT. L. REV. 75, 77, 100-03 (2002) (noting that "[c]ommercial software developers suffer because they have difficulty discerning how open source licensed software may affect their intellectual property" and proposing the creation of an open source license organization (OSLO) dedicated to maintaining and improving "buggy" open source licenses).

[24] When I use the term "proprietary software" in this Article I mean either a software product or other licensed software whose licensor requires royalties for use of the software, either in an up-front payment or on an ongoing basis. In addition, the term means software with narrow permissions for use and for which the source code is not disclosed or made available to the end users in the typical license transaction. In essence, I use the term to refer to the historically dominant practices used by software providers to protect and license their computer programs. However, some proprietary software allows some disclosure of the source code, particularly in the situation where the software vendor is selling software to other developers. In these cases, the usual restriction is that the customer-developer, while she has access to the source, may not distribute the source code to her customers. Microsoft has even launched its own "Shared Source Initiative . . . allowing customers, partners, governments, and competitors" access to its source code. *See* Microsoft, Shared Source Initiative Frequently Asked Questions, *at* http://www.microsoft.com/resources/sharedsource/Initiative/FAQ.mspx (Feb. 1, 2004). However, it does not want that practice to be confused with "open-sourcing." *Id.*

> OSS offers the significant benefits of community building, customer feedback, code transparency, and custom application development. The Shared Source Initiative

Such coexistence is essential for the long-term growth and viability of the open source movement. It is unlikely that the entire software industry will convert to the open source approach. Licensing coexistence relates to functional coexistence when open source and proprietary software must interoperate. Thus, open source software's licensing stance toward propriety software is critically important.

There are an increasing number of similar open source licenses, but they vary substantially in implementation. GNU/Linux uses the GPL, a license authored by the organization that creates the "GNU" part of the GNU/Linux operating system.[25] The GPL is an "infectious" license in the sense that I describe below. As such, its infectious terms are my foil to argue that such terms do not benefit the open source movement. The GPL is the legal foundation for GNU/Linux, an increasingly ubiquitous and critical component of the internet. Thus, while formally a private law instrument, its pervasiveness takes on a quasi-public character, further elevating the importance of its terms.[26]

An infectious license contemplates extending its terms to other software that is intermixed or coupled with open source software. The legal basis for this is potentially two-fold. The infectious license scope arises either from a

> seeks to afford these benefits while preserving valuable intellectual property rights in software.

Id.

[25] GPL, *supra* note 8; *see also* Zittrain, *supra* note 6, at 269 (comparing the GPL to the license for the Unix variant BSD, which has some elements of the GPL but does not include infectious terms).

[26] In labeling the GPL as "quasi-public," I draw upon the traditional distinction between the public law categories, such as constitutional, criminal and regulatory law, and the laws governing private arrangements and interactions, such as contracts, property and torts. Morton J. Horwitz, *The History of the Public/Private Distinction*, 130 U. PA. L. REV. 1423, 1424 (1982); see also Duncan Kennedy, *Stages of the Decline of the Private/Public Distinction*, 130 U. PA. L. REV. 1349, 1354-57 (1982) (arguing that the public/private distinction has been "loopified," and the line between public and private has become so blurred that the distinction is no longer workable). Given the importance of GNU/Linux to the internet, its code has a public character. Further, code can regulate in ways analogous to law's ability to regulate. LAWRENCE LESSIG, CODE AND OTHER LAWS OF CYBERSPACE 50-53 (1999).

broad conception of software derivative works,[27] or from expansive conditions on the permission to violate the reproduction right, such as permission to copy and use the software, or from hybrid effects of both. Both of these legal bases, however, are uncertain due to a variety of issues in copyright infringement of computer program source code.[28] Added to this uncertainty are questions about the status of open source licenses as contracts. Often this status question cannot be determined in the abstract when one considers only the license instrument itself.[29]

Some commentators posit that infectious terms intend to convert proprietary software into open source software.[30] Whether this theory is true

[27] *See* McGowan, *Legal Aspects*, *supra* note 7, at 5, 16-27 (explaining that "A might argue that if executing B's program caused A's program to be copied and to interact with B's program, then the combination of the two programs amounted to a work based on A's program, and therefore to a derivative work under the GPL").

[28] In latter sections of this Article I catalogue a number of factors that contribute to uncertainty for copyright infringement of computer software. These factors fall into three categories: (i) jurisdictional variety for the doctrines determining reproduction right and derivative work right infringement, and latent, linguistic ambiguity in the formulation of such doctrines, including such issues as insufficient specification of their scope of coverage; (ii) inconsistency in the application of these doctrines to specific disputes within jurisdictions, or across jurisdictions when multiple Federal Circuit Courts of Appeals have adopted the same doctrine; and (iii) indeterminacy in characterizing and finding the specialized facts underlying these technology-intensive disputes. While these three categories are obviously interrelated, their aggregate effect is to make many case outcomes at least partly inestimable in this area of rapidly advancing technology. *See* MELVILLE B. NIMMER & DAVID NIMMER, NIMMER ON COPYRIGHT § 13.03[A][1][d], at 13-51 (2003) [hereinafter NIMMER] ("[T]he uncertainty created by the ad hoc nature of the software cases . . . hampers development and progress in the computer software field. Software developers have no adequate guidelines regarding what level of independent development is required to avoid copyright infringement.").

[29] *Cf.* McGowan, *Legal Aspects*, *supra* note 7, at 9-16 (discussing a number of issues for the GPL, including questions of formation, issues of privity, revocability, assignment, term, and notice of the license provisions, and positing that for some of these issues the GPL might be enforceable as a contract depending on the provision to be enforced and the facts of the case).

In addition, these questions about open source licenses suggest alternative motivations for user adoption of particular licenses. *See* McGowan, *SCO What?*, *supra* note 3, at 33-34 (suggesting that the GPL terms may not necessarily be optimal for the developers who use them, but are employed in a trademark sense as a quasi-brand identity espousing certain development procedures or ideological beliefs developers may find more important than the terms themselves).

[30] *See* Christian H. Nadan, *Open Source Licensing: Virus or Virtue?*, 10 TEX. INTELL. PROP. L.J. 349, 359 (2002) ("Thus, if you incorporate some GPL code in your proprietary software product, arguably your whole proprietary product becomes open source and must be licensed by you under the GPL."); *see also* Daniel Lyons, *Linux's Hit Men*, FORBES, Oct. 14,

or not, infectious terms create disincentives against closely coupling and intermixing open source and proprietary software. Sometimes close coupling and intermixing will be desirable. Despite this fact, proprietary software vendors may implement defensive licensing provisions to guard against infectious open source terms, as might be expected from the rule of action and reaction.[31] For example, a proprietary license might prohibit distribution of the software in intimate combination with software whose license requires source code availability.

Moreover, it is not clear that infectious terms are necessary to support the growth of open source software. Apart from any debate about infectious licensing terms, as a privately-provisioned public good, open source software is an increasingly valuable part of the worldwide public and private computing infrastructure. It has entered markets dominated by proprietary, fee-based software and developed growing niches. It has done this without any dependence on, and perhaps due to its lack of, prices for ongoing use, although most open source software licenses allow distributors to charge for distribution and other complementary items. Thus, software application markets with open source products are mixed markets. Within these markets, infectious terms diminish opportunities to aggregate or combine open source and proprietary software in cases where it would be beneficial

2003 (discussing accusations that computer code licensed under the GPL could be used "to creep into commercial products so it can . . . force open those products"), *available at* http://www.forbes.com/2003/10/14/cz_dl_1014linksys.html. *But cf.* Zittrain, *supra* note 6, at 269 (contending that the central aim of the GPL is to prevent the "proprietization" of derivative software).

[31] For example, some vendors of proprietary software have implemented license agreements that explicitly prohibit intermixing proprietary software with open source software. *See, e.g.*, Master End-User License Agreement for Microsoft Software, *at* http://msdn.microsoft.com/subscriptions/downloads/EULA_MSDN_Jan03.pdf (last visited Dec. 20, 2004). Section 3.2.2 of this license states that it prohibits distributing the software or creating derivative works of the software if doing so would require the software to be: (i) disclosed or distributed in source code form; (ii) licensed for the purpose of making derivative works; or (iii) redistributable at no charge. *Id.* at 4.

Besides directing marketing efforts against open source software, proprietary software vendors might be expected to ensure that resources under their control are not deployed or deployable to assist the open source movement. Further, one might expect political efforts to emerge to counter the mounting interest in open source software by governments worldwide, some of which perceive open source as a way to lower information technology costs, while others see it as a technology transfer and local-ownership opportunity.

and where market conditions would allow prices to reflect principally the value component from the proprietary software.

To develop these themes, I proceed as follows. Part II reviews open source software licensing and the paradigmatic infectious terms of the GPL. My emphasis is to convey a general understanding of the prevalent licensing approaches for open source software. Then, I sketch how far-flung developers collaborate to create such software. Against this background, I present the GPL's infectious terms and discuss how they operate, or might be thought to operate.

Part III describes the technological framework for infectious scope. This part provides a technical description of software's ability to intermix and couple with other software. Programmers can modify and extend a program's capabilities in many ways. These details impact whether a license's infectious terms will reach other software. For example, in GNU/Linux, combinations of user programs and the operating system do not violate the GPL-based infectious license as applied in the Linux operating system kernel. But other combinations, within the Linux kernel component of GNU/Linux, may run afoul of infectious terms. Some understanding of the continuum of ways to modify, extend, intermix or couple software programs is necessary to understand the issues raised by infectious terms.

Next, Part IV examines the GPL's infectious terms within the continuum developed in Part III. Both the technological continuum and the GPL's infectious terms post minimal boundaries. So, Part IV also relates how some better boundaries have come about from practical implementations of the GPL and from a companion license that sheds light on the GPL: the Lesser GPL.

Part V develops and applies a descriptive model that isolates infectious terms and relates their potential legal bases to the technological framework for infectious scope. The legal bases which might anchor infectious terms include copyright's distribution and reproduction right, and derivative work right. Copyright's derivative work right is perhaps the most ill-defined and hard to qualify of copyright's exclusive rights.[32] Case law reflects

[32] *See* NIMMER, *supra* note 28, § 3.03[C][3], at 3-22.3 (discussing a number of paradigms for derivative works, and then noting that the "foregoing discussion ventilates a bewildering number of options for conceptualizing derivative works"); Lydia Pallas Loren, *The Changing Nature of Derivative Works in the Face of New Technologies*, 4 J. SMALL & EMERGING BUS. L. 57, 84-93 (2000) (arguing that copyright's derivative work right should not prohibit creation by others of integrated digital works that electronically reference, but do not copy, the original digital work).

uncertainty as to whether a software work is a derivative of an earlier work.[33] Complicating the situation is the practical and epistemological question as to whether the derivative work right is redundant given the reproduction right's application to non-literal infringement.[34]

Infectious terms seek to apply an open source licensing scheme to "other" software. The question then becomes: what "other" software do these terms reach? One possible answer is that their reach is coextensive with copyright's derivative work right, assuming that the derivative work right reaches more broadly than the reproduction right. If infectious terms are based on, and coextensive with, the derivative work right, then they are uncertain in their reach due to the vagaries of the scope of the derivative work right. The second possible basis is that infectious terms reach as far as a licensor's power to condition reproduction and use of a work. The interface between copyright, contract, and licensing law, also creates an area of uncertainty, particularly for licenses such as most open source licenses, where there typically is no explicit assent to the terms of the license.

Thus, uncertainty is the common denominator among the possible legal bases underpinning infectious terms. The uncertainty has costs of its own, but it also increases the trouble infectious terms create for software interoperability. Both of these impact the efficacy of infectious terms, which in turn influence the incentive effects arising from them.

Finally, Part VI concludes by emphasizing the implications of these themes.

[33] *Compare* Micro Star v. FormGen, Inc., 154 F.3d 1107, 1111-13 (9th Cir. 1998) (determining that FormGen's distribution of player-created "Duke Nukem" game levels could constitute infringement of the derivative work right because the audiovisual displays were embodied in concrete form in the game's MAP files), *with* Lewis Galoob Toys, Inc. v. Nintendo of Am., Inc., 964 F.2d 965, 967-69 (9th Cir. 1992) (determining that Lewis Galoob's "Game Genie," which allowed players to speed up and otherwise alter the output from a Nintendo game console, would not constitute a derivate work because the displays did not assume a permanent form). *See also* Montgomery v. Noga, 168 F.3d 1282, 1290-93 (11th Cir. 1999) (finding that the software was a derivative work because testimony indicated that anywhere from seventy to one hundred percent of the original code remained unchanged); United States v. Manzer, 69 F.3d 222, 227 (8th Cir. 1995) (finding testimony that "the computer files sold . . . were more than seventy-percent similar to the copyrighted software" was sufficient to uphold a finding that a derivative work was created).

[34] Loren, *supra* note 32, at 63-64; *see also* NIMMER, *supra* note 28, § 8.09[A], at 8-138.18 ("This right [the adaptation right] may be thought to be completely superfluous . . . .").

II. OPEN SOURCE SOFTWARE AND INFECTIOUS LICENSE TERMS

The David and Goliath story of GNU/Linux versus Microsoft's Windows juggernaut is only one facet of the startling open source software story. We do not know how the modern tale will end. Will the penguin beat the giant in the market, or will acrimonious co-existence reign? Regardless, many would agree that the open source movement has irreversibly changed the world of software. Open source has computer scientists puzzling about whether distributed ad hoc software development teams can or do collaboratively outperform traditional models for software development.[35] It has economists investigating why people volunteer their time on evenings and weekends to write and contribute source code to thousands of open source projects.[36] It has business academics questioning whether sustainable business models and profit opportunities are possible when valuable software is given away for free and those who charge for aggregating and distributing such software have little formal control over their primary production input.[37]

[35] This literature is growing too quickly to survey in a single footnote. For some paradigmatic examples, see Jamie Dinkelacker & Pankaj K. Garg, Corporate Source: Applying Open Source Concepts to a Corporate Environment 1 (2001) (Position Paper at ACM Workshop: Making Sense of the Bazaar: First Workshop on Open Source Software Engineering) (describing the application of "Open Source concepts, perspectives and methodologies within the corporate environment" including its debugging success); Yutaka Yamauchi et al., Collaboration with Lean Media: How Open-Source Software Succeeds 329 (2000) (Paper at ACM Conference on Computer Supported Cooperative Work) (describing that although software development has traditionally been a "coordination-intensive process" recent interest and advances in computer supported cooperative work facilitated the geographically distributed nature of much open source software development).

[36] The economics literature on open source software is also growing extensively. *See* Justin P. Johnson, *Open Source Software: Private Provision of a Public Good*, 11 J. ECON. & MGMT. STRATEGY 637, 637-62 (2002) (applying game theory to model open source software development as the private provision of a public good); David P. Myatt & Chris Wallace, *Equilibrium Selection and Public-Good Provision: The Development of Open-Source Software*, 18 OXFORD REV. ECON. POL'Y 446, 448 (2002) (exploring collective action problems such as free-riding and coordination costs in the context of open source software development); Josh Lerner & Jean Tirole, *The Simple Economics of Open Source* 35 (Mar. 2000) (discussing the motivations behind participation in open source projects, and inviting further study on the subject), *available at* http://papers.nber.org/papers/w7600.pdf.

[37] *See* Robert Young, *Giving It Away: How Red Hat Software Stumbled Across a New Economic Model and Helped Improve an Industry*, *in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION, *supra* note 2, at 113-14 (describing the genesis of Red Hat and how the company founders discovered the growing popularity of the emerging Linux operating system in the early 1990s, relating that: "When we'd ask where this Linux stuff was

Most facets of the story, however, focus on a license. The open source software license is the foundation for this new software production method, this new economic phenomenon for highly-leveraged volunteerism, and this new, strange, business opportunity.

The next section explains open source licensing and the innovative software development approach it enables. This is given as general background for a subsequent discussion of the GNU/Linux license, the GPL.[38] The GPL is one of the most widely used open source licenses. It also expresses the infectious terms that are the focus of my inquiry. Subsequent sections focus on the GPL specifically and explain the potential mode and scope of its infectious terms.

A. Open Source Licensing and Software Development

The open source licensing approach has common elements that most, but not all, open source software licenses implement. Open source software projects originate under one of many available open source licenses.[39] This

coming from, we'd get answers like, 'It's from the programmers according to their skill to the users according to their needs.'").

[38] A typical distribution of the GNU/Linux operating system employs other licenses in addition to the GPL because other components in the distribution are under such licenses. Thus, while the GNU "part" of the distribution and the Linux kernel "part" are under the GPL, there are other components in most distributions under other licenses. However, in GNU/Linux, measured by the lines of source code covered by particular licenses, the GPL dominates: one commentator's analysis rates the GPL as the license designated for 50.36% of the source code in the Red Hat GNU/Linux 7.1 distribution. David A. Wheeler, *More than a Gigabuck: Estimating GNU/Linux's Size*, *at* http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html (last modified July 29, 2002) (analyzing the Red Hat Linux distribution along the metric of source lines of code (SLOC), which, using automated software tools, in essence counts the lines of source code for each of the various components in the Red Hat Linux distribution, and in doing so notes the license applied for each component).

[39] There are a number of web sites listing open source licenses. *See* The Approved Licenses, *at* http://opensource.org/licenses (last visited Oct. 21, 2004). These licenses are listed by the Open Source Initiative (OSI), a certification system for identifying licenses that meet the certification mark's definition of "open source." *Id.*; *see* Various Licenses and Comments about Them, *at* http://www.fsf.org/licenses/license-list.html (last visited Oct. 21, 2004).

Due to the growing number of licenses, and the numerous other issues that corporate information technology departments face in deploying open source software, a company named Black Duck Software recently formed to offer software to help these endeavors. *See* Black Duck: Leading Solutions for Software Compliance Management, *at* http://www.blackducksoftware.com (last visited Dec. 20, 2004) (noting that the company provides an automated solution for open source code detection, license validation and means that the programmer or programmers who start a project typically select (or, more rarely, write) a license early in the project's life cycle. In choosing a license, the programmers adopt a copyright-based licensing regime that governs the software and grants conditional rights to others.[40] Thus, at bottom, open source software relies on the copyright status of computer programs as literary works, but deploys the rights arising from that status in a unique way.

The common approach among open source licenses is a conditional permission to violate one or more of the rights available under copyright. Thus, for example, a user may be allowed to modify the software. The license excuses any resulting violation of the derivative work or reproduction

management, and software auditing and certification). Black Duck's system has to be aware of the various licenses in order to provide the scanning, filtering, and automated intellectual property management capabilities it offers. Thus, their materials note the following: "[A]s of January 2004, the Open Source Initiative (OSI) has certified 47 licenses that comply with the Open Source Definition. (See http://opensource.org/licenses for the most up to date list of OSI approved licenses.) More than 20 additional uncertified Open Source licenses are also in common use."

White Paper: Black Duck Software: Solutions for Software IP Risk Management 4, *available at* http://www.blackducksoftware.com/wprequest.php (last visited Mar. 6, 2004) (downloaded and on file with author).

[40] It is unclear to what extent programmers are aware of their license choice in the specific. They may know that they need to place their project under a license. *See* SourceForge.net, Project: Gaim: Summary, *at* http://sourceforge.net/projects/gaim (last visited Dec. 20, 2004) (showing the project description for an open source program, one field of which is for designation of the specific license to govern the project).

Whether programmers shop for a specific license, read reviews or analysis of licenses, or merely pick one they have heard about is unknown. *See, e.g.*, Berkman Center for Internet & Society at Harvard Law School, Open Code – House of Licenses, *at* http://cyber.law.harvard.edu/openlaw/licenses (last visited Dec. 20, 2004) (listing "some of the current free, open, community, public, copyleft and right" licenses); Electronic Frontier Foundation (EFF), Guide to Licenses, *at* http://www.eff.org/IP/Open_licenses/licenseguide.html (last visited Dec. 20, 2004) (listing and describing a variety of open source licenses for documentation, music, and software); The GNU Operating System - Free Software Foundation, Licenses, *at* http://www.gnu.org/licenses/licenses.html (last visited Dec. 20, 2004) (listing and evaluating various open source licenses in terms of compatibility with the GPL); Open Source Initiative (OSI), Licensing, *at* http://www.opensource.org/licenses (last visited Dec. 20, 2004) (listing open source licenses in compliance with the "open source definition").

McGowan posits that programmers choose licenses for a trademark-like effect to communicate a social identity for the project rather than to have a specific legal effect. *See* McGowan, *SCO What?*, *supra* note 3, at 33-34 (suggesting that the GPL is employed in a trademark sense).

right, provided that the user meets the conditions specified in the license. Thus far, this is unremarkable since this is the basic mechanism underlying all licensing. What makes open source licensing unique is that the licenses permit a broad scope of activity and attach only a few conditions.[41] This inverts the typical approach of a proprietary software license, which permits only a narrow range of use and attaches conditions with considerably different consequences.

In open source software, the conditions signal the goals, reach, and effect of the license.[42] The licenses express a governance continuum over the user's activities. They implement varying levels of control, but all are less restrictive than typical proprietary software licenses. Thus, open source licenses stratify on a continuum. I elaborate in the subsection below, describing the typical license-condition groupings in order of lesser to greater scope. Then, I relate these groupings to open source software development norms and practices.

[41] This approach is often referred to as "copyleft." *See* Ira V. Heffan, *Copyleft: Licensing Collaborative Works in the Digital Age*, 49 STAN. L. REV. 1487, 1491 (1997) (describing "copyleft" under the GNU General Public License as an "agreement that permits reproduction and distribution of their [programmer's] works, but does not allow anyone to place further restrictions on them"); The GNU Operating System - Free Software Foundation, Licenses, *at* http://www.gnu.org/licenses/licenses.html (last visited Dec. 20, 2004) ("*Copyleft* is a general method for making a program free software and requiring all modified and extended versions of the program to be free software as well.").

[42] Because there is some uncertainty as to whether open source licenses will be enforced as contracts, at their least common denominator the written instrument will function as a copyright permission. The potential for contract enforcement, however, is not irrelevant to the potential outcome, and can have a particularly important impact on the remedies available to the licensor. *See* Sun Microsystems, Inc. v. Microsoft Corp., 188 F.3d 1115, 1122 (9th Cir. 1999) (holding that whether Sun could enjoin Microsoft based on a license agreement from Sun to Microsoft for Sun's Java technology depended on whether the license provisions in question were "license restrictions or separate [contractual] covenants."). In *Sun Microsystems*, the licensor, Sun, sought to have injunctive copyrights remedies apply. *Id.* Microsoft's position would have been improved in the dispute if the conditions at issue were contractual promises because in that case the default remedy is damages. This would increase Sun's requirements of proof, and give it less potential leverage over Microsoft.

1. Conditioned Permissions for Copyright Protected Software

Being "free" and "open" are often-described goals of open source software. Nonetheless, some prominent open source products have licenses that enforce neither goal, but require only attribution.[43] These licenses allow others to do practically anything with the software, including incorporation into proprietary software, as long as there is notice that the software originated from the open source project. These licenses do not even require that the source code be available – a key norm of the open source movement. Thus, attribution-only licenses are the least restrictive type of licenses used for open source projects.[44]

Most open source licenses impose more conditions than attribution-only licenses. They impose additional conditions on the user's permission to use,

[43] The most prominent example of a free software product using this type of license is the Apache web server software. Its license requires attribution in the form of an acknowledgment stating: "'This product includes software developed by the Apache Software Foundation (http://www.apache.org/).'" Apache Software License, Version 1.1, *at* http://www.apache.org/LICENSE-1.1 (last visited on Nov. 11, 2003) [hereinafter Apache License].

The Apache license was adapted from what is known as the Berkley Software Distribution (BSD) license, used to release the source code of a "flavor" of the Unix operating system developed at Berkley. *See* Marshall Kirk McKusick, *Twenty Years of Berkeley Unix: From AT&T-Owned to Freely Redistributable*, *in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION, *supra* note 2, at 42-46 (describing the initial decision to offer the entire BSD Unix flavor under the BSD license, due to the popularity of a networking component earlier offered under the license, and discussing a later, related lawsuit that pitted the other major flavor of Unix, at the time owned by AT&T, against the free BSD Unix distribution, the dispute being whether a small number of copyrighted AT&T components were present in the kernel of the BSD Unix).

The BSD style license requires attribution indirectly by requiring persistency of the copyright notice, which includes the name of the original author. *See* The BSD License (requiring redistributions in both source code and binary form to reproduce the copyright notice), *available at* http://www.opensource.org/licenses/bsd-license.php (last visited Nov. 12, 2003); *see also* NetBSD's Copyright and Licensing Terms, §§ 1, 2 (same), *available at* http://www.netbsd.org/Goals/redistribution.html (last visited Nov. 30, 2003); Sendmail License §§ 2, 3 (same), *available at* ftp://ftp.sendmail.org/pub/sendmail/LICENSE (last visited Nov. 11, 2003); The FreeBSD Copyright, §§ 1, 2 (same), *available at* http://www.freebsd.org/copyright/freebsd-license.html (last visited Nov. 30, 2003).

[44] Given that attribution-only licenses do not require that the software be free of royalties, or that source code be available, there is some question as to whether attribution-only licenses are properly called free or open source software. They are often categorized this way, however, because the programmers manage these projects using freely available source code and internet-based collaborative development.

modify, and redistribute the software. The permission to use is effectively synonymous with the permission to copy and run the software,[45] which practically all licenses grant. Thus, one can freely download a copy of most open source software and use it in modified or unmodified form.[46] Minimal conditions attach to this activity.

[45] Open source software raises a particular version of a doctrinal puzzle springing from 17 U.S.C. § 117 (2000). The puzzle is that § 117 allows an owner of a copy of a computer program to make an additional copy for statutorily defined reasons, including if the copy "is created as an essential step in the utilization of the computer program in conjunction with a machine and that it is used in no other manner." *Id.* § 117(a)(1). The puzzle occurs because with open source software, it is difficult to determine under the licenses and common practices whether each licensee is also the owner of a copy, in particular when physical media (e.g., a CD-ROM) is not employed to distribute the software. Traditional proprietary software licenses sought to characterize the transaction as a license that did not transfer ownership of a copy. *See* Michael J. Madison, *Reconstructing the Software License*, 35 LOY. U. CHI. L.J. 275, 314-15 & n.137 (2003) (discussing the right of an owner of a copy to run the program, and noting that: "Limiting that right to the *owner* of the program meant, under the strictest reading of the statute, that software companies that *merely licensed* individual copies of their programs to customers could avoid having those copies made subject to the Copyright Act.") (second emphasis added). While one of the original reasons for § 117 was to allow users to make backup copies of programs, it has become more important since courts have adopted the notion that a "RAM copy" is a violation of copyright's reproduction right. *See* Joseph P. Liu, *Owning Digital Copies: Copyright Law and the Incidents of Copy Ownership*, 42 WM. & MARY L. REV. 1245, 1256-63 (2001) (describing the genesis and evolution of the RAM copies doctrine, and noting that in the case spawning that doctrine the user's argument based on § 117 failed because the court found the user to be a mere licensee, and not an owner of a copy).

This impacts open source because the licenses typically do not contain the provision found in proprietary licenses characterizing the transaction as not involving any transfer of ownership of a copy. On the other hand, it is not clear from open source licenses that there is such a transfer. *See generally* Lothar Determann & Andrew Coan, *Spoiled Code?:* SCO v. Linux: *A Case Study in the Implications of Upstream Intellectual Property Disputes for Software End Users*, CYBERSPACE LAW. Jan. 2004 (noting that the GPL does not seem to have any characteristics of a sale of ownership of a copy). *But see* Eben Moglen, *Questioning SCO: A Hard Look at Nebulous Claims* 3-4 (2003), *at* http://www.osdl.org/docs/osdl_eben_moglen_position_paper.pdf (last visited Aug. 4, 2003) (noting that copyright law "contains a special limitation on the exclusive right to copy with respect to software," and arguing that users who have copies of software can execute the copies on one machine, but not explicitly addressing whether open source licensees in fact have "copies" as contemplated by § 117). *See generally* McGowan, *SCO What?*, *supra* note 3, at 18-20, 22, 31 (describing and analyzing the issue of § 117 copies in the context of the *SCO* case).

[46] In this context, the software is modified if someone changes the source code.

When the user redistributes the software in modified or unmodified form, conditions of greater consequence attach.[47] First, the redistributors must make the source code available.[48] Second, the redistributors may not charge royalties for use,[49] although aggregators and redistributors are typically allowed to charge for the distribution service and other services,

[47] Joseph Scott Miller, *Allchin's Folly: Exploding Some Myths About Open Source Software*, 20 CARDOZO ARTS & ENT. L.J. 491, 496-97 (2002) (noting that the "sharing requirement" under the GPL applies only to works distributed or published). *See generally* Henry W. Jones, III, *Impacts of Open Source Software on Your M&A Deal* (describing considerations raised by open source software use in the event of a sale of a company or a sale of its assets), *available at* http://www.softwarebusinessonline.com/newsletter2_march04.htm#feature2 (last visited Oct. 3, 2004).

[48] *See, e.g.*, GPL, *supra* note 8, preamble ("For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code."); Mozilla Public License Version 1.1, § 3.2 [hereinafter Mozilla Public License] ("Any Modification which You create or to which You contribute must be made available in Source Code form . . . ."), *available at* http://www.mozilla.org/MPL/MPL-1.1.html (last visited Nov. 11, 2003); Nokia Open Source License Version 1.0a, § 3.2 [hereinafter Nokia Open Source License] (same), *available at* http://www.opensource.org/licenses/nokia.php (last visited Nov. 30, 2003); Sun Public License Version 1.0, § 3.2 [hereinafter Sun Public License] (same), *available at* http://www.opensource.org/licenses/sunpublic.php (last visited Nov. 30, 2003).

The Open Source Definition (OSD) provides that "[t]he program must include source code." Open Source Definition, § 2 [hereinafter OSD], *available at* http://www.opensource.org/docs/definition.php (last visited Dec. 20, 2004). The OSD, however, is not a license, but rather a specification for a certification program operated by the Open Source Initiative to classify open source licenses. *See* Open Source Initiative (OSI), *at* http://www.opensource.org (last visited June 4, 2003) ("Open Source Initiative (OSI) is a non-profit corporation dedicated to managing and promoting the Open Source Definition for the good of the community, specifically through the OSI Certified Open Source Software certification mark and program.").

[49] For example, the GPL and the Perl Artistic License contain provisions disallowing royalties, but allow fees for various services such as distribution, support and separately provided warranties. GPL, *supra* note 8, § 1 ("You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee."); Perl Artistic License, § 5 (prohibiting fees for the software itself, but allowing a "reasonable copying fee for any distribution" and fees charged for support), *available at* http://www.perl.com/pub/a/language/misc/Artistic.html (last visited Dec. 20, 2004).

Other licenses use language granting "a world-wide, royalty-free, non-exclusive license." Mozilla Public License, *supra* note 48, § 2.1; RealNetworks Public Source License Version 1.0, § 2, *available at* http://www.helixcommunity.org/content/rpsl.html (last visited Nov. 30, 2003); Sun Public License, *supra* note 48, § 2.1.

The OSD provides that "[t]he license shall not require a royalty or other fee." OSD, *supra* note 48, § 1.

such as warranty and customer support.[50] Third, a redistributor must follow conditions relating to reapplication of the license terms to the redistributor's licensees when it modifies and distributes the modified software.[51] This reapplication provision requires further discussion.

Consider two possibilities for the reapplication provision. An open source license might say: when you distribute your modified version of the software, you must reapply to your licensees all of the exact same license terms under which you took the software. This is the strong scope of the provision. The weak scope relaxes the condition. It does not require the redistributor to reapply every license term, but delineates those that must be reapplied. For example, the license might require redistributors to reapply key provisions such as source code availability or the prohibition on charging royalties, but allow other license terms to escape reapplication. A relaxed reapplication provision creates a more flexible open source license. The strong scope of the provision ensures that the original license's terms "run with the code," even as the software evolves through the process of collaborative development and distribution.[52]

Open source licenses impose other conditions beyond the three I have discussed thus far. These include the "infectious" terms I will discuss in detail below, and other provisions less central and relevant to my argument. A few examples from the latter group are provisions attempting to deal with

[50] For GNU/Linux there are several companies that distribute the software and provide associated services. The leading and most well know example is Red Hat. It aggregates open source software components from a variety of sources, including some of its own open source software, to produce a GNU/Linux based "distribution" of the operating system. Red Hat Linux 9 Web Page, *at* http://www.redhat.com/software/linux/personal (last visited May 31, 2003) (downloaded and on file with author). The product description leads with: "Red Hat Linux 9 combines the latest Linux technology from the Open Source community in one easy to use operating system." *Id.* Red Hat charges for these distributions. Indeed, its GNU/Linux distribution has historically been available as "off-the-shelf" software in most computer retail stores.

[51] *See, e.g.*, GPL, *supra* note 8, § 2(b) (requiring that distributed modifications of GPL protected code be licensed under the GPL).

[52] *See* McGowan, *Legal Implications*, *supra* note 4, at 244-45 (discussing the use of copyright-based open source licenses to try "to create de facto property rights" in the software).

the threat of software patents,[53] provisions prohibiting discriminatory licensing,[54] and provisions disclaiming warranties and indemnification.[55]

While important, these non-central provisions are peripheral to the fundamental behavior driving the open source movement: source code availability for freely sharable and modifiable software. This behavior is enforced to some degree by open source licenses, but also takes significant conforming power from norms prevalent within the community of open source software developers, distributors, and users.

[53] A common provision in open source software licenses is a termination clause providing for the termination of rights if the licensee "initiate[s] litigation by asserting a patent infringement claim." Mozilla Public License, *supra* note 48, § 8.2; Nokia Open Source License, *supra* note 48, § 8.2; Sun Public License, *supra* note 48, § 8.2; *see also* IBM Public License Version 1.0, § 7 ("If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed."), *available at* http://www.opensource.org/licenses/ibmpl.php (last visited Oct. 20, 2004); RealNetworks Public Source License, *supra* note 49, § 11.1(c) (providing for automatic termination if the licensee at any time "commence[s] an action for patent infringement against Licensor.").

[54] The OSD prohibits open source licenses from discriminating against persons, groups or fields of endeavor. OSD, *supra* note 48, §§ 5, 6. Many of the licenses referenced in footnotes 43, 48, 49, and 53 through 55 are certified by the Open Source Initiative to comply with this requirement. *See* Open Source Initiative, The Approved Licenses, *at* http://www.opensource.org/licenses/index.php (last visited Oct. 20, 2004) (listing OSI certified licenses); Open Source Initiative, Certification Mark and Process, *at* http://www.opensource.org/docs/certification_mark.php#approval (last visited Oct. 20, 2004) ("The OSI Certified mark is OSI's way of certifying that the license under which the software is distributed conforms to the OSD . . . .").

[55] The OSD does not require open source licenses to include express or implied warranties, and most, if not all, licenses contain standard "no warranty" provisions and liability disclaimers. *See* OSD, *supra* note 48 (remaining silent on the issues of warranty and liability disclaimers); *see also* Apache License, *supra* note 43 (no warranty); GPL, *supra* note 8, §§ 11, 12 (no warranty and liability disclaimer); IBM Public License, *supra* note 53, §§ 5, 6 (same); Mozilla Public License, *supra* note 48, §§ 7, 9 (same); Nokia Open Source License, *supra* note 48, §§ 7, 9 (same); Perl Artistic License, *supra* note 49, § 9 (no warranty); RealNetworks Public Source License, *supra* note 49, §§ 8, 9 (no warranty and liability disclaimer); Sendmail License, *supra* note 43, § 6 (same); Sun Public License, *supra* note 48, §§ 7, 9 (same).

2. Open Source Development Norms and Practices

Open source software’s central goal, freely sharable and modifiable source code, is more than a legal mechanism implemented in a license. It is a powerful community norm that guides and facilitates the efforts of most large open source software projects.[56] Indeed, for open source projects based on attribution-only licenses, the combination of these norms and market appeal for the open source product suffices to ward off successful privatization of the software even though the attribution-only license itself does not prevent this.[57]

For example, another critical internet component is a well-known open source product called Apache, which serves web pages over the internet. Apache uses an attribution-only license.[58] Anyone could take the entire open source product and try to sell a proprietary version. Among other reasons, this does not happen because the proprietary version would not benefit from the continuous improvements to Apache arising from users and contributing programmers constantly submitting fixes for software bugs and submitting coding changes for the product. The proprietary version could not harness this effort arising from open source norms of sharing source code modifications. Thus, open source norms, sometimes strongly supported by the license, sometimes weakly supported, promote a unique collaborative software development effort.[59]

[56] Vetter, *supra* note 7, at 568 n.9, 604 n.112; *see also* Zittrain, *supra* note 6, at 272-73 (comparing developmental differences between free and proprietary software).

[57] Commandeering or privatizing an open source software project is a theoretical concern with a number of intriguing aspects, but in practice, it rarely occurs. This is in part due to the reason discussed in the text – superior demand for the open source version of the software due both to the success of that version, and the reputation of the programming group responsible for its development. Another prohibiting factor is that attribution-only licenses are not the dominant form of open source license.

A related phenomenon is forking, where a group of programmers takes an open source software project in a different direction, effectively creating two competing (or complimentary) open source products where before there was only one. Open source licenses allow forking, as long as the software continues to meet the conditions of the original license regime. Forking, like privatization, is also rare, but the possibility of forking is thought to provide an important disciplining force on the ad hoc leadership group in charge of most open source products. McGowan, *Legal Implications*, *supra* note 4, at 263-64.

[58] Apache License, *supra* note 43.

[59] The open source software collaborative development model is unique in comparison to traditional proprietary software development models. *See* Vetter, *supra* note 7,

Most large open source software projects, including Apache and GNU/Linux, operate using this collaborative development model. A core development group generates a substantial portion of the software. Other non-core developers and users operate and debug the software. The leaders of the project make design decisions and filter software submittals for inclusion in the product. The users and non-core developers, because they can review the source code, are better able to determine whether software problems lie in their own applications or in the open source software itself. This efficiency, among others, supports the open source norm of freely sharing modifications.

Specifically, the norm suggests to users and programmers that they share modifications with the core developers leading the open source project. Their confidence in doing this arises from at least two sources. First, for products under open source licenses that guard against privatization (by using terms stronger than an attribution-only license), the license ensures that others will not exploit the contributed modifications for private gain. Second, users' confidence in the open source process comes from the knowledge that even if someone attempted privatization, the original open source developers are better positioned than the privatized version to evolve the product to meet users' needs, satisfy market demand, and thus lead the field in the application's niche.

To see how open source development practices interact with a typical open source license, consider how an open source project might proceed. Assume that the core developers each program components of the software that are not easily partitioned or separately useful and are designed to interact as an operable whole. Putting aside the possibility of joint authorship,[60] each

at 626 & n.186 (noting that the means of collaboration among open source programmers are different in degree than programmers in traditional software development). The degree to which open source software is unique is too substantial to fully describe here, other than to provide a few examples. With the source code viewable by all, open source development is thought to have debugging efficiencies exceeding those of traditional software. Open source developers are typically far-flung and they collaborate over the internet. There is a lack of physical proximity among key developers in comparison to traditional software projects. Moreover, while having some partially-centralized decision-making apparatus, they lack the full command and control management apparatus used to control costs and track inputs and outcomes for proprietary development. *See id.* at 567, 597, 610 n.133, 625-29 & n.186 (comparing the traditional coordination of team developed software in the proprietary model with coordination of open source collaborative model).

[60] By designating the programmers as co-contributors, I consciously sidestep the question whether the co-contributors are joint authors in the copyright sense, or whether the resulting product is a joint work. *See* NIMMER, *supra* note 28, § 6.03 (discussing the elements

author has licensed her software to the others, and to any users, under the open source license. A web of license interdependency has begun.

If the core group accepts significant modifications from users/contributors, the interdependency web expands. Typically everyone eventually upgrades to the latest version containing the new modifications. As this process continues, it effectively locks the project into an open source development mode due to collective action constraints. It is unlikely all users/contributors with copyrighted code in the project would agree to privatize the project.[61] Even if most of the developers agreed to privatize, doing so would require an excision of the hold-out programmers' code. Sometimes such an excision will be very difficult because the hold-out programmers' code might be highly intertwined with the other programmers' code.[62] Thus, as an open source project grows, the license and development practices mutually reinforce the open source nature of the software.

of joint authorship and the distinction between joint authors and a joint work). Nimmer also contrasts joint authorship with its alternatives, derivative works and collective works:

> [I]n the case of both a derivative work, and a collective work, the contributing author owns only his own contribution, while in the case of a joint work each contributing author owns an undivided interest in the combination of contributions. What, then, distinguishes a derivative work from a joint work based upon inseparable parts? What distinguishes a collective work from a joint work based upon interdependent parts? The distinction lies in the intent of each contributing author at the time his contribution is written. If his work is written "with the intention that [his] contribution . . . be merged into inseparable or interdependent parts of a unitary whole" then the merger of his contribution with that of others creates a joint work. If such intention occurs only after the work has been written, then the merger results in a derivative or collective work.

Id. § 6.05 (quoting 17 U.S.C. § 101 (2000)) (footnotes omitted).

[61] Even if all contributing programmers privatized a project, this would only affect future versions of the software. Existing and past versions made available under an open source license would still be in circulation among users, who could rely on the original license terms for their continued use.

[62] Excising code from a large project, however, is a possible resuscitation method for GNU/Linux should SCO prevail in its suit against IBM. *See supra* notes 17-18 and accompanying text for a description of the *SCO* suit.

If SCO proved that IBM copied SCO's source code into the Linux kernel, the open source community has stated that it would then rework the Linux kernel to remove the code. *See* Stephen Shankland, *SCO: Unix Code Copied into Linux* (May 1, 2003), *at* http://news.com.com/2102-1016_3-999371.html (quoting Bruce Perens, a leader within the open source community, as stating that "a simpler solution" would be to allow the Linux community to replace the alleged infringing code). While excising the code could create an infringement-free version for the future, SCO would have the existing GNU/Linux user base

The preceding discussion illustrates that the open source creed, freely sharable and modifiable source code, starts with a license that establishes baseline or foundation norms, and builds from that license through development practices that reinforce the foundation. The open source license permissions that enable this foundation typically require availability of the source code, prohibit charging royalties, and require reapplication of at least some of the same terms to redistributed software. In a multi-contributor open source project, these license terms eventually create an interdependent web of permissions. The paradigmatic example of this reinforcing cycle is the General Public License and the open source software based on it. A discussion of this example follows in the next section.

B. The General Public License (GPL)

The General Public License, or GPL, is at the heart of the open source movement in many ways. To many, it is synonymous with open source software. Much of the academic commentary in a variety of disciplines focuses on the GPL as the paradigmatic open source license. It is the first and most widely applied open source software license to go beyond an attribution-only license.[63] The most important components of the GNU/Linux operating system are licensed under the GPL. Besides the GPL-based Linux kernel, the critical software tools from the GNU open source project rely on the GPL. This is because Richard Stallman, the author of the GPL, also leads the GNU project under the auspices of the Free

in a vulnerable position. Users would have to seriously consider paying licensing fees for the current version until they could upgrade to the resuscitated version, or face the possibility of paying copyright infringement damages to SCO for both past and future use.

[63] *See* Steve H. Lee, *Open Source Software Licensing* 15-19 (1999) (discussing the role of Richard Stallman and the Free Software Foundation in the genesis of copyleft and the GPL, "the first example of an open source software license"), *available at* http://cyber.law.harvard.edu/openlaw/gpl.pdf (Apr. 28, 1999); Josh Lerner & Jean Tirole, *The Scope of Open Source Licensing* 23 (Dec. 2002) (noting "the dominant role of the General Public License" after a survey of the popular Sourceforge.net open source project repository revealed that approximately three-quarters of the projects used the GPL, as compared with approximately seven percent under the BSD), *available at* http://papers.nber.org/papers/w9363.pdf; The GNU Operating System - Free Software Foundation, Licenses, *at* http://www.gnu.org/licenses (last visited Oct. 28, 2004) (noting that the GNU GPL is used by over half of all Free Software projects).

Software Foundation (FSF).[64] Mr. Stallman is a pioneer and visionary of the open source movement, which further raises the visibility of the GPL. Thus, for a variety of reasons, the GPL is the benchmark for open source licenses.[65]

The GPL implements the open source license conditions described above. Its preamble explicitly recites what has become the general goal of the open source movement: in contravention to proprietary software, the GPL "is intended to guarantee your freedom to share and change free software - to make sure the software is free for all its users."[66] It requires

[64] *See* Free Software Foundation, *at* http://www.fsf.org/fsf/fsf.html (last visited Oct. 28, 2004) (describing the foundation as "dedicated to promoting computer users' right to use, study, copy, modify, and redistribute computer programs"); *see also* McGowan, *SCO What?*, *supra* note 3, at 35 (suggesting that the FSF owns any cognizable intellectual property rights, such as copyright or trademark, to the GPL itself).

[65] Among the other reasons why the GPL is a standard-setter is that the GNU organization, the FSF, maintains a listing that critiques other open source licenses as compared to the GPL. *See* GNU Operating System - Free Software Foundation, Various Licenses and Comments about Them, GNU Project - Free Software Foundation, *at* http://www.gnu.org/licenses/license-list.html (last visited Oct. 28, 2004) (listing, categorizing and discussing licenses based on whether they are compatible with the GPL). Given the leadership position of the FSF, and its progenitor, Mr. Stallman, in the open source community, these critiques have significant effect on others in the community. *See id.* (describing another open source license, the Mozilla Public License (MPL), as incompatible with the GPL and urging programmers not to use the MPL for that reason, but noting that a later version of the MPL allows the licensor the option to specify other license terms for part of the MPL'ed code, and if that other license is GPL-compatible, then combinations with that part of the program would be allowed); *accord* Mozilla Relicensing FAQ (noting that the FSF has asserted that the MPL is incompatible with the GPL), *available at* http://www.mozilla.org/MPL/relicensing-faq.html (last visited Oct. 28, 2004).

[66] GPL, *supra* note 8, preamble.

The GPL's preamble also comments on two of the license terms earlier described as less central and relevant to my argument, but important generally to open source software: software warranty and patent issues. *See supra* Part II.A.1.

> Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.
>
> Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

GPL, *supra* note 8, preamble.

that source code be available,[67] and that those redistributing the software in modified or unmodified form do so without charging royalties for use.[68] Users can run a GPL-based program without restriction. The license is mostly concerned with copying, distribution, and modification of the software.[69] The GPL provides that users retain their rights to copy, modify, and distribute the software, even if the parties from whom they obtained the software violate the GPL and lose their rights.[70]

The GPL specifies a strong scope for the reapplication provision, requiring redistribution under the GPL's terms.[71] In addition, it further specifies that redistributors may not add to the GPL's conditions.[72]

> Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. *You may not impose any further restrictions on the recipients' exercise of the rights granted herein.*[73]

The GPL specifically acknowledges that its licensing power derives from copyright in the software.[74] The copyright protection attaches to subsequent copies of the software, providing the aggregate group[75] of contributing

[67] GPL, *supra* note 8, at §§ 1, 3.

[68] *Id.* §§ 2(b), 11.

[69] *Id.* § 0.

[70] *Id.* § 4.

[71] The GPL implements a strong scope for the reapplication provision in two places. For redistribution of verbatim copies, section 1 applies. *Id.* § 1. It requires that such a redistributor "conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program." *Id.* For non-verbatim copies, the redistributor "must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License." *Id.* § 2(b).

[72] *Id.* § 6.

[73] *Id.* (emphasis added).

[74] *Id.* preamble, § 5.

[75] It is important to emphasize that this aggregate contributing group usually grows over time in a large open source software project. Users become contributing programmers when they submit bug fixes. New programmers join the project, or junior programmers cycle through the project. Assuming that the project does not require copyright assignments to some centralized authority, each new contributor expands the web of permissions that underlie and authorize open source promulgation of the software. *See supra* Part II.A.2.

programmer-authors[76] with the power to exclude uses that do not comply with the GPL's conditions. In this sense, and because it is based on copyright, the GPL "run[s] with the code."[77] Accordingly, it does not necessarily need to rely on contract to hold licensees to the conditions underlying the permission. Typically, there is no formal act of assent to the GPL's terms, and the GPL does not contemplate such an act.[78] In this scenario, those who redistribute GPL-based software typically do not

[76] Concerning the programming group's power to exclude others' use, as before, I put aside issues of joint authorship. These issues, however, might have dramatic effects. If the open source software were adjudged a joint work, then "the conclusion of joint authorship follows, meaning that both parties were free to exploit the entire product." NIMMER, *supra* note 28, § 6.05, at 6-15. Thus, at least under copyright, a sole joint programmer-author might be able to license the entire open source software project to others on different terms. Arguments that there was an agreement, or estoppel, or other theories of equity might constrain a rogue joint author, but with potentially much less effectiveness than the open source license's imposition of the interdependency web. The web constrains the programmer-authors and their users because each has authored only a component that requires the rest for full operation. McGowan, *Legal Aspects*, *supra* note 7, at 13-14.

[77] McGowan, *Legal Implications*, *supra* note 4, at 245.

[78] Absent a formal act of assent, the GPL raises enforceability questions similar to those surrounding shrinkwrap licenses. *See* Patrick K. Bobko, *Linux and General Public Licenses: Can Copyright Keep "Open Source" Software Free?*, 28 AM. INTELL. PROP. L. ASS'N Q.J. 81, 100-03 (2000); Steve H. Lee, *Open Source Software Licensing* 72-79 (Apr. 28, 1999) (pre-publication version), *available at* http://cyber.law.harvard.edu/openlaw/gpl.pdf (last visited Dec. 20, 2003); McGowan, *Legal Implications*, *supra* note 4, at 289-96.

Beyond shrinkwrap licenses, their cousin, clickwrap licenses, with the benefit of the user's positive indication of assent to the terms, are another possible comparison point for open source licenses. However, open source software is not as inclined as proprietary software or web sites to present users an opportunity to affirm terms of the license. *See* OSD, *supra* note 48, § 10 (requiring that "[n]o provision of the license may be predicated on any individual technology or style of interface" because section ten "is aimed specifically at licenses which require an explicit gesture of assent in order to establish a contract between licensor and licensee. Provisions mandating so-called 'click-wrap' may conflict with important methods of software distribution . . . such provisions may also hinder code re-use."). The current version of the GPL was written in 1991, before clickwrap practices were prominently used. As a result, the GPL relies on the more traditional, shrinkwrap style "notice-plus-conduct model." McGowan, *Legal Implications*, *supra* note 4, at 289. For a more expansive discussion of shrinkwrap and clickwrap agreements and their implications for the digital era, see Michael J. Madison, *Legal-Ware: Contract and Copyright in the Digital Age*, 67 FORDHAM L. REV. 1025, 1031-43, 1055-76 (1998).

explicitly agree, in a contract sense, to reapply the same provisions.[79] Perhaps to compensate for this, the quoted language above expresses an imputed intent: all original and subsequent licensors who contributed to the software also grant a license to all possible end users who might receive the software via potentially innumerable redistributors.

The other implication of redistributors not expressly agreeing to the GPL is that this may raise factual questions about whether they intended the GPL to apply to software they distributed along with, or coupled to, the original GPL-based software. An example in this vein is the *SCO* case. Recall that the essence of SCO's suit against IBM is that IBM had improperly injected SCO's code into certain versions of GNU/Linux. Besides suing IBM, SCO began soliciting license fees from end users of GNU/Linux.[80] To allege that such users needed to pay license fees, SCO, in essence, had to allege that distributors, such as Red Hat, were distributing infringing copies of SCO's code. In response, Red Hat filed a declaratory judgment action.[81] One of the allegations in Red Hat's complaint is that SCO itself distributed GNU/Linux, and thus should comply with the license.[82] This would prohibit SCO from henceforth soliciting license fees from users.[83] A number of factual findings will ultimately determine the effectiveness of this argument.[84] Whatever the

[79] GPL, *supra* note 8, § 5 ("You are not required to accept this License, since you have not signed it.").

[80] *See supra* note 18 and accompanying text.

[81] In counts one and two of the complaint, Red Hat requested a declaratory judgment with respect to SCO's copyright and trade secret assertions. Complaint filed by Red Hat, Inc. at 19-20, ¶¶ 70-77, Red Hat, Inc. v. SCO Group, Inc., (D. Del. 2003) (No. 03-772) [hereinafter Red Hat Complaint], *available at* http://lwn.net/images/ns/rh-complaint.pdf (last visited Dec. 20, 2004). Red Hat also brought claims against SCO based on the "unfair, untrue, and deceptive campaign" being waged by SCO in order "to create an atmosphere of fear, uncertainty and doubt about LINUX." *Id.* at 1, ¶ 1. Red Hat's claims included false advertising and deceptive trade practices under the Lanham Act, and common law unfair competition, tortious interference with prospective business opportunities, and trade libel and disparagement claims. *Id.* at 20-23, ¶¶ 87-105.

[82] *Id.* at 1-2, ¶ 2.

[83] *Id.* at 19, ¶¶ 72-73.

[84] The factual findings underlying whether SCO's distribution of GNU/Linux is a defense for users, and the basis for a declaratory judgment for Red Hat include, assuming that SCO's original allegation against IBM is true, whether SCO knew that its code was included (if it was) in the GNU/Linux distribution(s) that passed through SCO. As of October, 2004, developments in the case include the court's denial of SCO's motion to dismiss and the court ordering the case stayed pending the resolution of the SCO versus IBM case in Utah. Red Hat, Inc. v. SCO Group, Inc., No. 03-772-SLR, at 1, 5 (D. Del. Apr. 6, 2004) (order denying

legal weight of these findings, they might have been strengthened by SCO's affirmative act of assent to be bound by the GPL.

Like other open source licenses, the GPL has additional provisions less central and relevant to my argument. These include provisions attempting to deal with the threat of software patents[85] provisions dealing with the lack of warranties and indemnification, and disclaimers of liability.[86]

As the pioneer license and most popular standard-setter, the GPL has widely influenced the open source software community.[87] Its basic thesis, source code availability for freely sharable and modifiable software, has been implemented in many other licenses. The power of this thesis rests in the software development efficiencies it creates for collaborative open source projects. The aggregate effect of this power enables Tux, the little Linux penguin, to viably compete with Microsoft in niche, but important, application markets. Despite its overall importance and influence, the "infectious" license terms of the GPL are less often imitated compared to its other core provisions.[88] While less imitated, they are oft discussed and create

SCO's motion to dismiss and order staying case but indicating that if the Utah litigation is "not progressing in an orderly and efficient fashion, the court may reconsider the stay"), *available at* http://sco.tuxrocks.com/Docs/RH/RH-34.pdf (last visited Dec. 20, 2004). The court also ordered parties to submit updates of the Utah litigation involving IBM by letter every ninety days. *Id.* at 5. Red Hat filed a motion for the court to reconsider its order to stay the case but the court has yet to rule on the motion. Motion for Reconsideration, *Red Hat* (No. 03-772-SLR), *available at* http://sco.tuxrocks.com/Docs/RH/RH-35.pdf (last visited Dec. 20, 2004).

[85] GPL, *supra* note 8, §§ 7-8. Section 7 admonishes the user not to redistribute the software if doing so contravenes the GPL's terms due to a court order or intellectual property infringement concerns. *Id.* § 7 ("For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.").

[86] *Id.* §§ 11-12. Section 11 provides an "as is," "as provided" disclaimer of warranty. *Id.* § 11. Section 12 disclaims liability. *Id.* § 12. The GPL provides no indemnification; no provisions of the license speak to indemnification.

[87] It seems likely that programmers choose the GPL because it is the first and most well-known license. The GPL also probably lures adopters because it is the license employed by GNU/Linux, the flagship open source software product. *See* McGowan, *SCO What?*, *supra* note 3, at 33-34 (suggesting that the GPL is employed in a trademark sense).

[88] Devon Bush, *Analysis of Prevalent Open-Source Licenses*, *at* http://cyber.law.harvard.edu/home/ossummary (last visited Dec. 20, 2004) (comparing seven of the most widely used open source licenses, and noting that the GPL "provides one of the strongest copylefts available" and is therefore "extremely viral").

another lightning rod for controversy in the debate about open source software development and its mode of legal protection. In the next section, I turn to the GPL's "infectious" terms to describe the provisions involved and the issues they engender.

C. The GPL's Infectious License Terms

The GPL speaks in several places about its scope of coverage for other software. It accounts for a number of situations where one redistributes something other than, or more than, a verbatim copy of some or all of the originally-received open source software. Most of the relevant provisions, however, are in section 2 of the GPL.[89] This provision contemplates two

[89] Given the importance of the GPL's section 2, it is set out below.

> 2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
>
> a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
>
> b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
>
> c) [requires a certain notice mechanism under certain conditions]
>
> These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it. [hereinafter § 2, ¶1]
>
> Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program. [hereinafter § 2, ¶2]
>
> In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License. [hereinafter § 2, ¶ 3]

GPL, *supra* note 8, § 2.

outcomes for the licensing status of this "other"[90] software: it is, or is not, required to be distributed under the GPL's terms when distributed along with the original software.

While GPL section 2 is clear that it envisions two possible outcomes, what is less clear is the path to either outcome. In the analysis below, this Article evaluates the section 2 labyrinth.[91] First, I parse and extract the relevant provisions. Next, I categorize them to isolate the areas of certainty and uncertainty. Each is discussed in turn. In all categories, the ultimate questions are: (i) what does the license mean when it says "other" software; and (ii) how does the GPL treat this "other" software.

At a length running a little over 400 words, consuming most of a page, section 2 contains the GPL's infectious terms.[92] It covers several situations that may arise when open source software is modified, intermixed or coupled with other software and redistributed.[93] Besides the introductory preamble paragraph, it has three labeled provisions, subsections (a) through (c), and three unlabeled paragraphs, which I refer to as paragraphs 1 through 3.[94] Some of these items are not related to my argument: subsection 2(a) requires notice within the source code when one modifies it;[95] and section 2(c) requires, in a certain technical scenario, notice related to copyright, warranty, and the applicability of the GPL.[96] Overall, however, even for the relevant

[90] It is my own construction to use the word "other" in this way: "other" indicates software that is not the originally-received open source software. However, GPL section 2 uses the word in a similar construction, reciting the "other work" (and "another work") when referring to other software. *Id.* § 2(b), ¶ 3.

[91] *See* Peter S. Menell, *Envisioning Copyright Law's Digital Future*, 46 N.Y.L. SCH. L. REV. 63, 181 (2002) (describing the GPL as a "a complex licensing agreement designed to prevent programmers building proprietary limitations into 'free' software").

[92] GPL, *supra* note 8, § 2.

[93] *Id.*

[94] *See generally id.*

[95] *Id.* § 2(a) ("You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.").

[96] Subsection 2(c) is as follows.

> If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

provisions, GPL section 2 is as significant for what it does not say as for what it does.

The GPL's infectious terms are distributed among subsection 2(b) and paragraphs 1 and 3. Subsection 2(b) requires that redistributors "must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge."[97]

Paragraph 1 elaborates on subsection 2(b), and perhaps extends it, while paragraph 3 provides a counter-example in which there is no infection.

From the copyright perspective, subsection 2(b) suggests a permission to violate the reproduction right, distribution right, and the derivative work right in the originally-received open source software.[98] By distributing or publishing a work that "contains" some or all of the original code, one would violate the reproduction right. Putting aside questions related to copyright's first sale doctrine[99] as applied to digital copies and the ownership thereof,[100] such activity would also violate the distribution right. Subsection 2(b) also subjects "derived" works to the GPL, implicating copyright's derivative work right.

The operative infectious language in subsection 2(b) is the middle clause, set off by commas: "that in whole or in part contains or is derived from the Program or any part thereof."[101] In combination with section 2's preamble and the rest of subsection 2(b), this language says that if I take some or all of the originally received software and distribute or publish it verbatim, or in modified or derived form, then I must do so under the GPL's terms, including licensing the distribution "as a whole at no charge."

Id. § 2(c).

[97] *Id.* § 2(b).

[98] *See id.*

[99] 17 U.S.C. § 109(a) (2000) ("Notwithstanding the provisions of section 106(3) [the distribution right], the owner of a particular copy . . . lawfully made under this title, or any person authorized by such owner, is entitled, without the authority of the copyright owner, to sell or otherwise dispose of the possession of that copy."); *see* R. Anthony Reese, *The First Sale Doctrine in the Era of Digital Networks*, 44 B.C. L. REV. 577, 579-80 (2003) (suggesting that new digital technology threatens to undermine copyright's first sale doctrine).

[100] *See* Liu, *supra* note 45, at 1266, 1274-78 (arguing that ownership of digital works should functionally provide similar incidents of ownership as physical copies of a copyrighted work provide, and arguing that the first sale doctrine for digital copies is greatly limited if not eliminated).

[101] GPL, *supra* note 8, § 2(b).

In the middle clause, the word "whole" is of particular interest. First, the same word appears near the beginning of subsection 2(b), but is used there in a different context than in the last clause. The earlier appearance is adjectival, describing one aspect of the originally received software (all of it) the use of which would require the license permissions specified. The subsequent appearance of "a whole" seems to identify the software the redistributor distributes. Whatever "a whole" includes is what the GPL's infectious terms cover.

Thus, "a whole" could be a literal copy of the originally received software. But, "a whole" could also be a slightly modified copy of the original source code. Further, "a whole" could be a highly modified copy of the original source code, in which the modifications include significant portions of other source code. The sequence could continue, defining increasingly expansive notions of "a whole" resulting from combination with "other" software. Thus, in part, the reach of the GPL's infectious terms depend on what is meant by "a whole." This conclusion, however, is not apparent from subsection 2(b). The reader must consider subsection 2(b) in the context of paragraph 1.

Subsection 2(b) introduces the concept of "a whole," but paragraph 1 significantly elaborates on it, as set forth below.

> These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. *But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.*[102]

This elaboration is the heart of the GPL's infectious terms. Given its importance, I further reference each of its components as sentences 1 through 3.

Sentence 1 notes that the requirements of section 2 apply to the new "whole."[103] I find in sentence 2 a three-part test, the identifiably independent

[102] *Id.* § 2, ¶ 1 (emphasis added).

[103] *Id.*

and separate works test, which I discuss below.[104] If the test is met, the other software is not part of the "whole" – and perhaps this adds some additional contour to the reach of the infectious terms. Sentence 3 elaborates on subsection 2(b)'s prescription that the "whole" reaches "the entire whole, and thus to each and every part regardless of who wrote it."[105]

This review of GPL section 2 shows that it takes an expansive approach to the issue of its coverage for other software. In this, "other software" could mean, on the one hand, minor modifications added to an existing open source program, which the GPL would capture. Or, on the other hand, completely separate software (possibly previously licensed as proprietary software) intermixed or coupled with the open source software.

The GPL has an expansive approach because section 2 stops somewhere just short of extending its terms to other software merely aggregated with the open source program. It draws this fuzzy boundary using the rubric of the "whole" – a label for a standard that is not expressly identified, but which may take its gist from the reproduction right or derivative work right in copyright.[106] As the GPL acknowledges, it needs this rubric to "control the distribution of derivative or collective works based on the Program."[107] In the GPL's view, whatever falls into the "whole" should be further licensed, when distributed as a part of the "whole," under the royalty-free, open source approach.

The problem GPL section 2 addresses is the allocation of rights or permissions for follow-on software works or for integrating disparate software to achieve a functional result for end users.[108] While this problem is not unique to open source software licenses, it is uniquely unwieldy in such software.[109] Follow-on developers need not have any connection to the

[104] *See infra* Part III.A (discussing the identifiably independent and separate works test of GPL section 2, paragraph 2).

[105] GPL, *supra* note 8, § 2, ¶ 1.

[106] In addition, the gist of the outer boundary of the "whole" may come in part from specific implementations of the GPL, from parole documents that inform its meaning, and from practices in the open source community. *See infra* Part IV.

[107] GPL, *supra* note 8, § 2, ¶ 2.

[108] *See* Mark A. Lemley, *The Economics of Improvement in Intellectual Property Law*, 75 TEX. L. REV. 989, 991 (1997) (discussing the changes in the "law of improvements," and noting that "[a] number of doctrines in modern copyright and patent law attempt to strike some balance between the rights of original developers and the rights of subsequent improvers").

[109] In the license contracts often employed in proprietary software development projects, vendors and customers employ a number of approaches to allocate follow-on rights.

original developers. And there is often no contractual relationship of the kind used in proprietary software development contracts to allocate or assign rights for follow-on development or integrated systems.[110] Often, follow-on open source developers entangle themselves with the prior development group. The allocation of rights, however, is an uncontested or moot point because the developer contributes the code to the project under an open source license.

Thus, in the context of an open source license's conditional permission to use, modify, and redistribute the software, infectious terms are most troubling when the open source software is intermixed or coupled with non-open-source software.[111] In these cases, the allocation of rights and

The rights may be allocated in advance to one party or the other. Another approach is to include in the contract procedures by which rights to improvements to preexisting software are awarded to one party or the other. Sometimes the contracted-for software development is an improvement to either the software vendor or customer's existing software, or improvements to both resulting in the integration of each with the other. Sometimes the specific improvements cannot be described in advance. Thus, a contract might designate a board or other group to determine on an improvement-by-improvement basis who takes the rights. In addition, joint ownership is an option. Finally, besides assignment of the rights, permissions or licenses for use within the scope of the project (or for other defined fields of use) may round out the allocations. *See generally* 1 ROGER M. MILGRIM, MILGRIM ON LICENSING 5A-28, 29 (1995) (discussing the desirability of using contractual provisions to better define the extent of rights to modifications given the "yet undefined scope of the 'derivative work' concept").

[110] There are exceptions to the estimation that most open source projects do not explicitly assign rights. Some open source projects request that contributors sign form agreements assigning or licensing copyright rights and privileges to an umbrella organization coordinating the project. *See, e.g.*, GNU Enterprise, Copyright Assignment, *at* http://www.gnu.org/software/gnue/community/copyleft.html (last visited Dec. 20, 2004) (requesting all developers contributing to "GNUe" code assign their copyright to the Free Software Foundation (FSF)); OpenOffice.org Open Source Project, Joint Copyright Assignment by Contributor to Sun Microsystems, Inc. ("Sun"), *at* http://www.openoffice.org/licenses/jca.pdf (last visited Dec. 20, 2004) (displaying the joint copyright assignment form all contributors to Sun's "OpenOffice.org" project are required to sign); *see also* McGowan, *Legal Aspects*, *supra* note 7, at 14-16 (discussing the possible implications of assigning copyright to organizations willing to police and enforce license terms, such as the FSF).

[111] A variation of the issue is when the open source software is intermixed or coupled with other open source software that employs a non-GPL-compatible license. The Free Software Foundation provides guidance listing licenses that it concludes are not GPL-compatible. GNU Operating System – Free Software Foundation, Various Licenses and Comments about Them, *at* http://www.fsf.org/licenses/license-list.html (last visited Dec. 20, 2004).

privileges is not moot. One who intermixes or couples open source software with her proprietary software risks a copyright infringement suit for violation of the reproduction and derivative work rights in the open source software.[112] This is the fundamental underlying mechanism of open source licensing, and indeed, software licensing in general.

The next part reviews various scenarios when other software falls under the GPL's rubric of the "whole." These scenarios are technical. I describe some of the common ways in which software is intermixed, combined, integrated, and coupled. In describing each scenario, the point is to relate the technical description to the GPL's rubric of the "whole." This is to analyze the effects of that rubric and to explore how infectious licensing terms might work. The GPL section 2 terms are my focus as a foil to explore more broadly the implications of the infectious approach.

The analysis in latter parts views the infectious approach skeptically. The next part illustrates the technical aspects of the approach. Even in light of the purported benefits that infectious terms support open source community development, prevent privatization, or convert proprietary software to open source, countervailing considerations arise from the technical scenarios that must be measured against the rights of copyright and the language of broad infectious terms. The variety of methods available to intermix and couple software creates legal uncertainty with attendant costs. Moreover, that same uncertainty has inhibiting effects for interoperability.

III. THE TECHNOLOGICAL FRAMEWORK FOR INFECTIOUS AMBIT

The previous part introduced open source software's licensing approach and development model. It also dissected GPL section 2's infectious terms into components. In the language of section 2, this is primarily an exercise in divining what "the whole" might mean in different situations. These situations exist in a technological continuum, in a framework that this part describes. In the forthcoming sections I do not yet analyze the framework

[112] Intermixing and distributing open source and proprietary software also raises infringement risk for the distribution right. However, in this context (and in most other contexts), violation of the distribution right follows from the violation of the reproduction right. *See* NIMMER, *supra* note 28, § 8.02[A], at 8-27 (noting "it is the act of copying that is essential to, and constitutes the very essence of all copyright infringement," including the distribution right). While a violation of the distribution right is also often a violation of the reproduction right, it does not necessarily follow that a violation of the reproduction right is also a violation of the distribution right. *Id*. § 8.02[C], at 8-30 (noting that "a printer who reproduces copies without then selling [or distributing] them is nonetheless an infringer").

scenarios against the legal basis underlying the GPL: copyright's reproduction right and derivative work right. Instead, that analysis comes in subsequent parts. Rather, the following analysis introduces some of the factual and technological scenarios that may fall into increasingly broad degrees of scope for the infectious terms.

A. Noninfectious Scenarios for Aggregated Software

GPL section 2 provides two linguistic formulations that delineate with some clarity, in terms of traditionally understood legal and technological concepts, when a redistributed work will not be "part of a whole" to which the GPL's terms aspire.[113] One formulation is paragraph 3, discussing mere aggregation.[114] The other is sentence 2 of paragraph 1, discussing identifiably independent and separate works.[115] In addition, paragraph 2 provides a general expression of intended goals for the infectious GPL terms.[116]

With regard to paragraph 3, mere aggregation for distribution purposes does not invoke the GPL's infectious terms. "[M]ere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License."[117] Testing this formulation in stages, a distributor who licenses or sells proprietary software and includes in the distribution media an installation kit[118] for an open source software

[113] GPL, *supra* note 8, § 2.

[114] *Id.* § 2, ¶ 3.

[115] *Id.* § 2, ¶ 1.

[116] *Id.* § 2, ¶ 2.

[117] *Id.* § 2, ¶ 3.

[118] I use the term installation kit to encompass a variety of technologies used to distribute the many files that comprise most software. For example, most installation kits include only a few files, with one special file that a user executes, or is automatically executed on certain events, such as inserting a CD into a computer. When the special "startup" file is executed, it extracts files from the other files, which are only containers holding the actual files that make up the software. Both source code and object code files could be extracted and installed. The installation kit's special startup file also will typically configure settings and parameters in the operating system of the target computer so that it recognizes the installed software and can run it. From the perspective of GPL section 2's terms, the significance in my example of the open source software being bound up in its own installation kit was to emphasize its separateness, which underscores that the overall distribution is only an aggregation.

product unrelated to the proprietary software is surely within the mere aggregation safe harbor.[119]

Does this conclusion change if the included installation kit is for software that will somehow interoperate or work with the proprietary software? Perhaps, but it depends on how. Currently there is no answer in the GPL, and only opinions in practice. The question is: where does "mere aggregation" stop and being "part of a [redistributed] whole" begin? This question frames the fundamental uncertainty clouding the GPL's infectious terms. However, the mere aggregation safe harbor immunizes a wide number of technical scenarios. It allows for flexibility in distribution media, and allays fears that merely installing GPL-based software on one's computer will invoke the infectious terms. Proprietary software does not become open source software just because one stores it on the same hard drive or CD-ROM as some GPL-licensed open source software.

There is a second, perhaps redundant, safe harbor in GPL section 2, the "identifiably independent and separate works" provision of paragraph 1, sentence 2.[120] This provision may be thought to give content to what is a "mere aggregation." It would seem that a separate work meeting the strictures of sentence 2 might also be a mere aggregation. Regardless, it provides a three-part test for when an identifiable section is not "a part of the whole": (i) the section is not derived from the originally-received open source software; (ii) it can be reasonably considered an independent and separate work; and (iii) it is distributed as a separate work.[121] The test intones both objective and subjective perspectives. Perhaps the objective aspect, whether one could reasonably consider a work independent and separate, justifies resorting to commercial practice to determine when "other" software is "part of a whole."[122] As the discussion in Part IV below

[119] My assertion that the proprietary software distributor is in the "mere aggregation" safe harbor assumes that the vendor has otherwise complied with the open source license terms. Thus, among other obligations, the open source installation kit should include source code files (or otherwise arrange for source code availability) and the distributor must not charge royalties for use. In addition, the co-distributed open source product must be licensed under the open source license, not the license that applies to the proprietary software.

[120] GPL, *supra* note 8, § 2, ¶ 1; *see supra* note 104 and accompanying text.

[121] GPL, supra note 8, § 2, ¶ 1.

[122] When the terms of a contract are ambiguous, the majority of jurisdictions allow extrinsic evidence, including evidence of trade usage, to aid in interpretation. 5 ARTHUR L. CORBIN, CORBIN ON CONTRACTS § 24.10 (rev. ed. 1998); *see also* U.C.C. § 2-202 (2002) (allowing evidence of trade usage, course of performance and course of dealing with or

explains, the GPL's broad provisions are sometimes implemented in practice to cover only particular technological methods for intermixing and coupling software.

Thus, the GPL provides two safe harbors, the mere aggregation provision, and the identifiably independent and separate work provision. Intermixed or commingled proprietary software must fall within one of these two in order to escape the infectious rubric of the "whole." In comparing the two safe harbors, the GPL's language specifies the mere aggregation safe harbor with a greater degree of specificity. Both, however, are intended by the license to remove other software from the "whole."

In the next section, I look at the other end of the spectrum. These are situations where the other software is highly likely to be a part of the "whole," meaning that the infectious license requires that the other software employ open source terms.

B. Modifications and/or Extensions to Source Code

While sentence 2 of paragraph 1 introduces a concept of identifiably independent and separate software sections, sentence 3 turns the concept around: if these "same sections" are distributed "as part of a whole which is a work based on the Program," the GPL's terms must apply to all such included separate sections, regardless of their authorship and previous licensing scheme.[123] In combination with section 2(b), this further specifies that the GPL's terms apply when "other" software is deemed to be "part of a [redistributed] whole."[124]

As with mere aggregation on one end of a continuum, certain ways of modifying or extending software will classify at the other end if the resulting work is based on the originally-received open source software, and is thus part of "a whole" when redistributed. In the language of copyright, these modifications or extensions are clearly or presumptively violations of the reproduction right or derivative work right, or both. In the rest of this subsection and the subsections below, I will discuss examples falling along this continuum.

To continue the discussion, I must introduce a few technological aspects of software. It has a source code form and, typically, an object code form,

without a showing of ambiguity); RESTATEMENT (SECOND) OF CONTRACTS § 202(1) (1981) (providing that all contracts should be "interpreted in the light of all the circumstances").

[123] GPL, *supra* note 8, § 2, ¶ 1.

[124] *Id.*

sometimes called "binary" form. Putting aside innumerable technical distinctions, programmers can understand, interpret, and manipulate the source code form, and use that form to write software.[125] The source code is written in a computer language, which somewhat resembles traditional human written languages, but has numerous unique "words" (language statements) for a variety of functions related to manipulating data, interacting with and commanding the computer, interacting with other software, controlling the flow and progression of the program, and performing other tasks in the computing environment.[126]

When a programmer completes her source code, she then processes or compiles it into object code form, which a computer can read and run.[127] The computer is able to read and run the object code because the computer has a special program called the operating system, which runs (or "executes") other programs and enables those programs to interact with the hardware.[128]

Since a computer program has a language that resembles a written human language, from a copyright perspective, software has drifted into classification as a literary work. Thus, similar to the way one might be liable for copyright infringement of the reproduction right if one copied this Article and rewrote half of it by minor paraphrasing, and left the rest verbatim, one might be liable for copyright infringement by copying the source code of a

[125] Vetter, *supra* note 7, at 582-83 (noting that programmers can understand, interpret and manipulate source code more productively than object code).

[126] *Id.* at 578-86 (noting the various functions of the source code and demonstrating these functions through a metaphor comparing software instructions to the instructions in a cooking recipe).

[127] *Id.* at 584. The copyright act's definition of a "computer program" reflects the technological distinction between source code and object code by recognizing that the program can command the computer to do work either indirectly, via the source code, or directly, via the object code: "A 'computer program' is a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result." 17 U.S.C. § 101 (2000); *see* Vetter, *supra* note 7, at 578-80 (describing a three-element model of computing, including: a first element, the source and object code forms of the computer program; a second element, the operating system, which is itself a special program that runs all the other programs; and a third element, the computing result, that is, the ongoing output and outcome of the various programs running and interacting in the operating system).

Often the programmer "finishes" the source code many times, meaning that upon compiling and running the software, she discovers problems. Thus, refining and finishing software is an iterative, trial-and-error process.

[128] Vetter, *supra* note 7, at 584.

sixty-page[129] computer program and reworking half of it by trivial reprogramming.[130] Both the resulting source code and object code would likely infringe the original. If the original source code was GPL protected open source software, the resulting software would be part of the "whole" and necessitate redistribution under the GPL. This is the minor modification example. The reproduction right for both literal and non-literal copying[131] is involved. The derivative work right may be involved as well.[132]

In the same way that someone could make increasingly drastic and extensive modifications to a copy of this Article, so could a programmer increasingly revise the sixty-page program. But even for these increasing

[129] I use the phrase "sixty-page computer program" only to convey a sense of the work's scope. The computer does not measure the program's length in pages, but perhaps by the size of the file that holds it. Programmers might size the program by the lines of code that it contains. And, if one printed it, it would paginate, and reveal its sixty-page length.

[130] In this example of a hypothetical sixty-page computer program, I should point out an oversimplification: the implicit assumption that there is copyright protectable expression in the program. Presumably, the program has "thin" copyright protection from improper literal misappropriation. Menell, *supra* note 91, at 65-66 ("Copyright law provides a thin layer of protection for computer software, effectively prohibiting wholesale piracy of computer programs without affording control for interface specifications and other essential elements of computer functionality."); *see, e.g.,* Eng'g Dynamics, Inc. v. Structural Software, Inc., 26 F.3d 1335, 1348 & n.15 (5th Cir. 1994) (noting that "'highly functional' software may lie very near the line of uncopyrightability").

For non-literal copyright infringement, however, the program may or may not have expression that survives the first two stages of the abstraction-filtration-comparison test of the Second Circuit's *Altai* decision. Computer Assocs. Int'l v. Altai, Inc., 982 F.2d 693, 706-10 (2d Cir. 1992) (announcing the test and applying the first two steps); *see* Menell, *supra* note 91, at 84 (noting that *Altai* has received a broad following).

[131] In cases involving non-literal infringement, courts have applied the "abstraction-filtration-comparison" test set forth in *Altai*, or some variation thereof. *See, e.g.*, Gates Rubber Co. v. Bando Chem. Indus., Ltd., 9 F.3d 823, 834-38 (10th Cir. 1993) (adopting the *Altai* approach, but specifically identifying six different levels of abstraction and six different elements subject to filtration). In literal infringement cases, however, where exact copying can be shown, courts are divided on whether to apply the same test. *Compare* Bateman v. Mnemonics, Inc., 79 F.3d 1532, 1544-46 (11th Cir. 1996) (finding that the *Altai* test applied in a case where literal infringement was shown), *with* Sony Computer Entm't, Inc. v. Connectix Corp., 48 F. Supp. 2d 1212, 1217 (N.D. Cal. 1999) (omitting the "filtration" step when the defendants admitted to copying), *rev'd on other grounds*, 203 F.3d 596 (9th Cir. 2000).

[132] The qualification as to whether the derivative work right is involved arises from the somewhat ambiguous nature of that copyright entitlement. *See* NIMMER, *supra* note 28, § 3.01 ("In a broad sense, almost all works are derivative works in that in some degree they are derived from preexisting works.").

revisions, the modified program is likely to be part of the "whole" because it will continue to be based on the original.[133]

In the modification examples, I have assumed without saying that the modifications are all new code. That is, they are new authorship and not copied from preexisting material. While this assumption is somewhat artificial, I use it to confine what I mean by a modification.

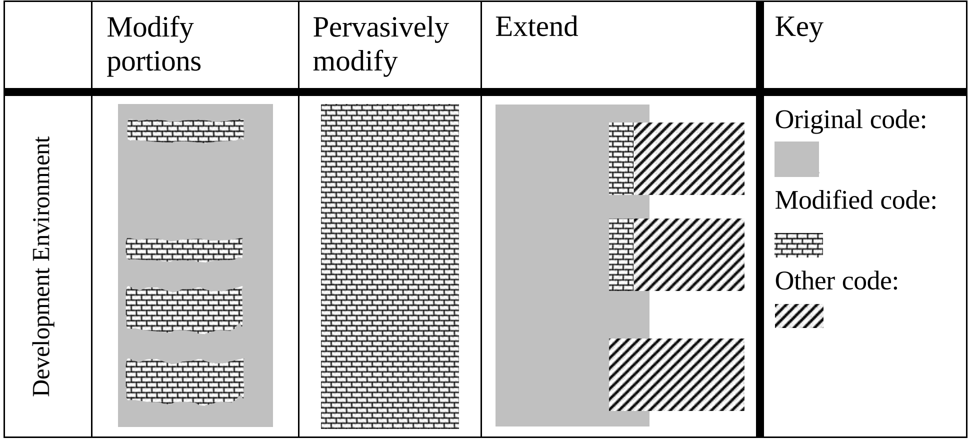
Often, however, programmers change and evolve code by inserting segments of preexisting code from a variety of sources. When this occurs while revising the originally-received open source software,[134] I call it extending the source code. The extension may be a verbatim insert of other source code, or it may be an insert followed by some revisions necessary to adopt the inserted code to the target program's operation.

Figure 1, below, depicts the different ways to modify and extend source code discussed above.

[133] The increasing revisions of the program might at some point eliminate all literal infringement, leaving only non-literal reproduction right infringement. Increasing revisions move the modified work into a gray zone overarching non-literal infringement and preparation of a derivative work. Loren, *supra* note 32, at 63-64. Classic derivative works of this Article would include a translation into Spanish, and a documentary film based on it. Similarly, a translation of the original program into another computer language would be a derivative work, as might an audio-visual training CD based on the program, using the program as an example training exercise.

[134] Under my description, which compares the computer program to this Article, modification and extension of the program are likely to result in new, "other" software being included in the "whole" for both the program's source and object code. The example is simple in that both this Article and the hypothetical program are thought to be managed as a single unit. That is, although it need not be this way, each is stored in a single computer file. The programmer makes the modifications or extensions to the file containing the source code, and compiles a new object file that the computer can execute. Many have experienced this simple situation when they first learned to write a program: one file of source code gets compiled into a single executable file, which, when executed, runs and performs a computing operation.

Figure 1 – Modifying and Extending Software



In Figure 1, the first two illustrations show code modified to a different extent. A real-world example of pervasive modifications would be translating a program from one programming language to another, perhaps from the C programming language to the Java language. The code extension illustration shows that one can add new code to a program, but might need to modify some of the original to fit the new code into the puzzle. All of these illustrations assume that the programmer develops the software in the development environment and generates a single executable to run the new program.

To discuss the next set of scenarios, I must complicate things with some abstractions and variations. In the simple situation, the file is just a container. To think of the software as confined to a single file (like my Article is confined to a single Microsoft Word file) is to place an artificial limit on how the code can be organized into containers, and on the types of containers usable. Things other then traditional computer files, such as databases, repositories, and source code control systems,[135] can hold the source code. These variations are concrete in the sense that they all name real mechanisms and capabilities available in the computing environment. They are used by programmers to organize, catalog, and manage code.

[135] A technical purist would probably observe that these “other things” that I note could hold code all rely on, or are held in, a file of some type recognized by the computer’s operating system.

To compare this to my Article, if it grows too long, I might decide to keep Part I in its own file, Part II in another file, and so on. Microsoft's Word product will let me work with all of these separate files as a "virtual" single document. In the ordinary sense of the word, they might be part of a "whole" document even though they are stored using a separate Word file for each part. The parts relate to and rely on each other, and each would be to some degree incomplete without the rest. Similarly, the hypothetical sixty-page program might be stored in logical subdivisions in multiple files, or multiple containers of other types. In effect, the programmer can "mix and match" containers holding code for a variety of pragmatic reasons.[136]

In considering the containers that hold code, recall that software has both a source code and object code form. This allows the mixing and matching to occur in and across both forms. For example, my hypothetical sixty-page program might be thirty pages of newly written source code joined with "thirty pages" of code I have in an object code file.[137] I can compile my thirty pages, and then join, or "link,"[138] it with the "thirty-page" object code file to create a whole program.

Variations from this simple example set the stage for the next section. The various ways to organize code into containers and then join or relate them in source code or object code form is just the first step in a greater description of the software development and execution environment. While the preceding section has suggested that modified or extended open source software is likely to be included in the infectious terms' conception of the "whole," this conclusion is less clear when intermixing and coupling code, which I discuss in the next two sections.

[136] Some typical reasons to segment code into multiple containers include: (i) increasing the ease of reusing the code in a segmented container; (ii) facilitating the process whereby multiple programmers work on the same program; and (iii) allowing incremental improvement to components of the program. Each of these reasons could allegorically apply to the various parts of my Article.

[137] The object code provides the executable form needed by the computer, so it is somewhat artificial to characterize it as "thirty pages" in length, but assume that the source code from which it is compiled was that long.

[138] The use of the word "link" to describe the operation of joining the two object files into an executable program file has a different (but analogous) meaning compared to today's internet parlance for the word "link" (the internet meaning follows a dictionary meaning for the word, indicating a reference from one item to another). Link, as applied to a programmer "linking" two object files in the development environment, typically results in a single executable file containing the contents of both object files. Thus, when a user runs the executable file in the computer's runtime environment, the program is complete. It has what it needs to achieve the desired computing result.

C. Intermixing Code in the Development and Runtime Environments

In the preceding section, I describe extending and modifying software as acts that occur in the development environment, rather than during the "runtime" environment. "Runtime" is, for our purposes, simply when the computer is running, and when the programs of interest are executing. In other words, the programmer works in her development environment writing code, assembling code from multiple sources, and generally planning to create a program that will function in the runtime environment. In this, the programmer can, and usually will, build her program to rely on functionality provided by other programs available, or invoke-able, in the runtime environment. The foremost example is the operating system. It creates and delivers the runtime environment. Every other program relies on the operating system to execute (i.e., run) the program, and to provide a variety of functions that the program can invoke to do "work" – that is, to achieve various computing results.

Thus, while I reserve the terms "modifying" and "extending" for the developer's activities when building the program, I apply the non-technical terms "intermingling" and "intermixing" as a general category descriptor for the various technological ways in which the programmer will write code that relies on other code at runtime. In this sense, every program, when executed by the operating system, is intermingled or intermixed with the operating system. Thus, the operating system is perhaps a special case of intermixing. This section will review some typical technological intermixing scenarios. Again, as before, the goal is to relate the technical scenario to the infectious terms in order to initially gauge whether the intermixing results in a "whole" that must be licensed under the GPL.

Before describing these scenarios, however, I must introduce an additional concept about the operating system's runtime environment: the notion of a "process."

Computer technologists have a special name for an executing program that the operating system is currently running. They call such a program a "process." A process is a program in operation or execution. Under the supervision of the operating system, the program is loaded into the computer's memory and shares memory and processor availability with other processes. The computer has a runtime environment created by the operating system. In this environment, a process is akin to an agent in the real world. The process can receive commands, carry them out, follow previously given instructions, interpret events to determine whether to follow conditional instructions given in the source code, access and manipulate data, communicate with other processes, take actions based on messages from other processes, become confused, err or create errors, and die.[139] One consequence of processes in an operating system is that they can be numerous. Many programs can be executing at once, each known to the operating system as a process. Processes will rely on and interact with the operating system, but they can also rely on and interact with other processes.

Recall the example of my hypothetical sixty-page program created by joining thirty pages of new code with "thirty pages" of code I have in an object code file. In the first example, I posited joining (called "linking") the two code segments in the development environment. The programmer, however, has an alternative. She could join the two code segments in the runtime environment. The technical term for this is "dynamic linking." Instead of joining, or statically linking, the two programs into a single executable file in the development environment, the programmer plans for the object code file to be available in the runtime environment. She writes her program to invoke or call functions in the object code file, without having those functions necessarily be in the executable container file that creates the process, or be wholly copied into the process itself. The object file that she will invoke at runtime may go by several names, depending on the operating system and technology involved. These names include "shared libraries," "libraries," or "dynamically linked libraries."

Both static and dynamic linking are examples of intermingling and intermixing code. The former intermixes in the development environment by linking or compiling the code into a single unit or container. Dynamic linking intermixes in the runtime environment; the executing process invokes or calls the code from another container (typically a file). The invoked code performs some expected function or work for the invoking program. This linked code is often called a "library." The differences between static and dynamic linking may have copyright implications. From the perspective of

[139] Another comparison that may be useful is to envision a computer program and process as analogous to a recipe. The recipe, which is like the source code, provides the sequence of instructions. Imagine that it is written in a language the kitchen's cook does not understand, so someone translates (compiles) it for that cook (operating system) so the cook can implement the recipe in the kitchen (computing hardware). Thus, while the cook is implementing the recipe, that is, while the dish is in progress, it is a "process" – a recipe being "run" or "executed." A cook implementing the recipe in the kitchen is like an operating system running a program (resulting in a process) in a computer. *See* Vetter, *supra* note 7, at 580-82 (describing in greater detail the recipe-cook-kitchen analogy for software and computing); *see also* Zittrain, *supra* note 6, at 271 (analogizing a computer program to a recipe in the context of decompiling object code into source code).

the linked, library object code file, assuming copyright protectable expression in that file, static linking triggers, in the development environment, the reproduction right in the library object code file. Static linking may also trigger the derivative work right. The static link operation creates a copy of some or all of the code in the linked object file.

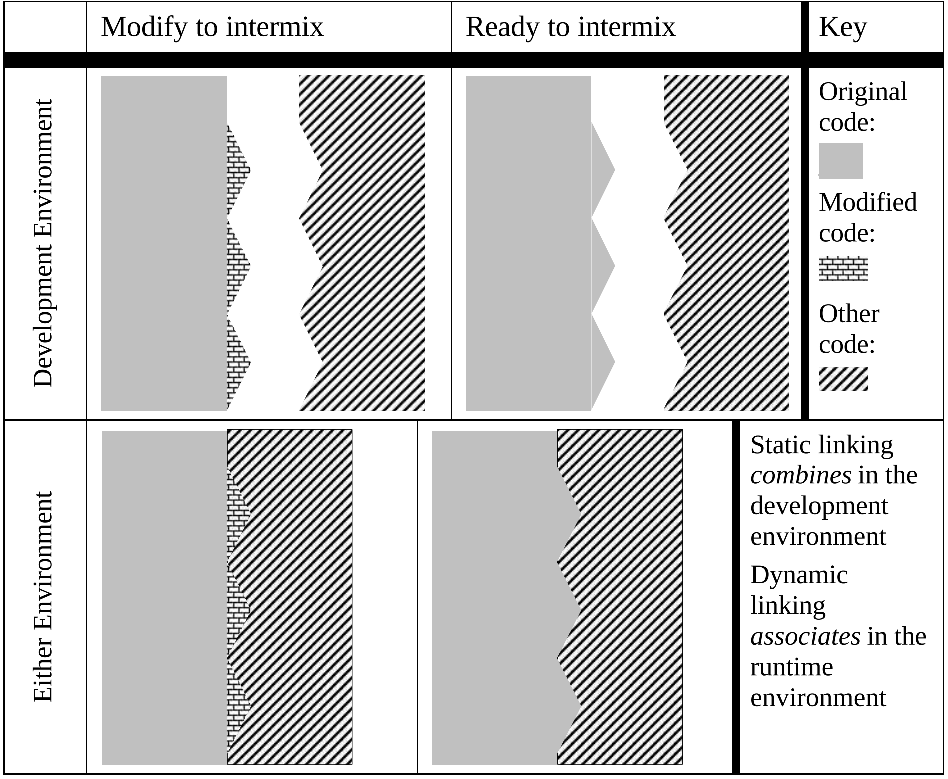
Dynamic linking, however, does not generally create copies of the library object code file in the development environment (although it may cause copy creation later at runtime because the invocation may load the library object code into memory, which under current copyright doctrine is a copy that violates the reproduction right). During development, the thirty-page source code program need merely refer to the functions it will later invoke, in runtime, in the object code file. It need not copy them into the executable file, as would a static link.

The situation changes for dynamic linking in the runtime environment. The calling program is executed, meaning that the operating system runs it as a process. Depending on how the programmer has determined to use and invoke it, the library object code file may be loaded into memory when the calling program is executed, or may be loaded later during the life of the process. Sometimes the library object code that the calling program expects to use is already loaded into memory because it is a "shared library." Whether the library object file is exclusively intended for use with the calling program, or is a shared library, the distinguishing feature is that the code executed by the process for part of its work is code that was not found in the original executable container file. That file contains only a reference to the library-located code (and perhaps contains some supporting technical information for the reference). Thus, while planned by the programmer during development, the intermingling and intermixing of the calling program and the library object file actually occur during runtime.

This section has explained a number of ways software can be intermixed by programmers, contrasting intermixing that occurs during development versus runtime. This intermixing occurs across multiple containers that might hold source or object code. By these two factors, (i) multiple containers, and (ii) the development versus runtime distinction, I distinguish intermingling and intermixing from the more simple ways of modifying and extending the code in a single file (container) discussed in the previous section. The simple modification and extension of code is analogous to creating, revising, and perhaps supplementing, a single Word document. Intermingling and intermixing code is like referencing, linking, and combining several Word documents into a greater whole, yet retaining the separateness of each document.

Figure 2, below, illustrates intermixing software in the development and runtime environments.

Figure 2 – Intermixing Software



Static linking *combines* in the development environment

Dynamic linking *associates* in the runtime environment

In Figure 2, the first illustration shows code that needs modification to intermix. Alternatively, some software is designed to intermix with other code, shown in the example in the second full column of Figure 2. In either case, both illustrations could fit together in one of two ways. First, the programmer could combine the code into a single file in the development environment by statically linking them into a single, executable object code file. Second, the original code and the new code could remain in separate files until runtime, where dynamic linking allows code from both files to run under the control of the process executing the program.

This section builds on the previous section that explained modifying and extending code. The next step is intermixing code, which can occur in either the development or the runtime environment. Intermixing code also shows that software can be arranged in multiple containers, usually files. These boundaries will impact the analysis as to whether infectious terms should reach "other" software combined with open source software. The next section adds the last level to this continuum by describing how separate processes (programs in execution) might interact.

D. Coupling and Integrating Software in the Runtime Environment

A program in execution is a process, and a process is like an agent in the real world.[140] The process is an agent interacting with other agent-processes, all under the watchful eye of the computer's operating system (which is itself a specially delineated collection of processes). Many software applications are in fact multiple agent-processes enacting cooperative processing or delivering a unified user experience. Together, they deliver functionality to the user, sharing work based on the programmer's design, which springs from the nature of the computing tasks involved in the application, and the technology platform for the computer, i.e., what operating system and other software technology is available.

These cooperating processes are coupled or integrated. This means that they are designed to work together interactively by sharing data and referencing each other, whether they comprise an identified application (such as a database, office productivity package, etc.) or whether they simply make use of each other's services. Some processes exist to serve. They come alive at some point, launched by the user or an event knowable by the computer, and then wait for other processes to ask them to serve. An example is a process (or group of processes) that couriers data from process to process according to a common protocol. Processes can interact through messaging or other mechanisms provided by the operating system. One process can cause another to come to life, ask it to do work, and terminate it when the work is complete.[141] An application might consist of several processes that all start at once and cooperatively deliver functionality, to end

[140] A process can receive commands and follow instructions, interpret events, manipulate data, communicate with other processes, err or create errors, and die. *See supra* Part III.C.

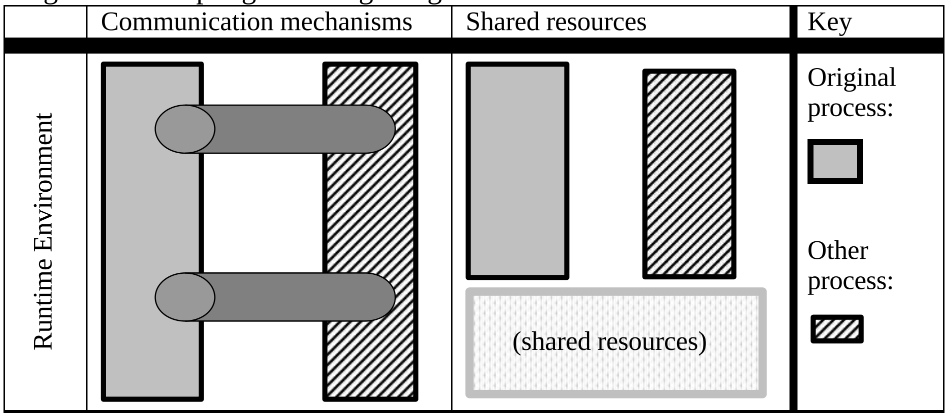
[141] Bringing a process to "life" by initiating a computer program to execute in the operating system usually involves loading executable code into memory. This is a violation of the reproduction right for any copyright protected code in the executable file. *See* Liu, *supra* note 45, at 1264-67 (noting that digital copies containing copyright protected content violate copyright's reproduction right under the RAM copies doctrine).

users, or perhaps as part of the inner workings of some computing resources such as a database or networking capability.

Within a single computer, processes represent the last concept needed to cover the potential configurations of software that might cause other software that is somehow connected with GPL protected code to fall under the GPL's definition of the "whole." Processes are one of the cornerstone design concepts underpinning current operating systems. Recalling the divisions discussed above,[142] programmers have tremendous flexibility when designing an application. They can locate the code in one or more container files, and link those files statically in the development environment, or deliver multiple executable files to the user and arrange for dynamic linking of executable code. Thus, a single process may draw from one or more code-containing files for its computer program instructions. This single process, however, might be merely one among many that comprise the application. Designers can arrange and configure an application among one or more processes, each arising from one or more code-containing files, depending on the desired design factors.[143]

Figure 3, below, illustrates coupling and integrating software in the runtime environment.

Figure 3 – Coupling and Integrating Software



[142] *See supra* Part III.C.

[143] Design factors to determine whether to use multiple processes in a software application include: (i) the type of application; (ii) the data it will store, use, or deliver; (iii) the sophistication of the end users; (iv) the operating system(s) upon which it will run; and (v) many other considerations.

In Figure 3, the illustrations suggest some of the typical means by which processes interact and integrate in the runtime environment. Each program, while in execution under the operating system, is a process. Like a person in the real world, the process can communicate with others, exchanging data or signaling status, interests, or other attributes. The first illustration shows this. The second illustration shows two processes using a shared resource, and thereby coupling or integrating. The shared resource could be computer memory, a file, or other resources found within the operating system.[144]

Before moving to the next section, where I will apply the GPL's terms in the context of this framework, I have summarized Part III in the following table:

Table 1 – Summary: The Technological Framework for Infectious Ambit

Category	Modify or Extend	Intermingle or Intermix	Couple or Integrate	Mere Aggregation
Description	Software in a single container or file, modified; or extended by copying code from external sources	Software in multiple containers, intended to function under the control of a single process, linked at development or runtime	Software in multiple processes, interacting and interdependent to some degree	Separate software on the same media

This framework describes the technological options that bear on whether the GPL's infectious "whole" requires the non-open source software to succumb to its terms. The framework, however, applies beyond the GPL to any open source software license. The GPL is only my foil; other licenses

[144] While I will not extend my framework further, it is even possible to have a software application comprised of multiple processes, but where the processes are located on different computers connected by a network. There is no particular need to extend my framework because, for the purposes of this Article, such a configuration is not so dissimilar from multiple processes operating on the same computer. From the software's perspective, it may use different mechanism(s) to communicate to other, remotely located processes or gain access to shared resources. Regardless, network connected processes are sufficiently similar to processes interacting on the same system to allow them to be considered together in the analysis of the GPL's "whole."

have infectious terms.[145] Licenses that implement infectious terms, or conversely, licenses that wish to provide a safe harbor from such terms, may wish to define a boundary somewhere along this framework.

Better boundaries for infectious scope reduce problems of uncertainty cost and chilling interoperability. In this, infectious terms may retain their beneficial aspects to support open source development and prevent privatization, yet shed problematic effects and disincentives that limit their efficacy to support the open source movement.

The next part discusses the GPL's approach to the problem, building on the earlier discussion of the GPL's conception of a "whole."

IV. THE INFECTIOUS GPL TERMS IN THIS FRAMEWORK

The GPL provides a reasonably certain safe harbor from its infectious terms for software merely aggregated with open source software, but appears to offer scant guidance for the scenarios of modifying, extending, intermixing, or coupling other software with the GPL protected open source code.

Section IIIA above analyzed the GPL's safe harbor for (i) aggregated or (ii) identifiably independent and separate software. Without repeating this analysis, both of these safe harbors are clearly intended by the license to remove other software from the "whole." They are likely to be effective for several reasons. First, such programs are outside the framework established above. They are not coupled and integrated, just as a music department is

[145] *See, e.g.*, Mozilla Public License, *supra* note 48, § 3.1 ("The Modifications which You create or to which You contribute are governed by the terms of this License . . . ."); Nokia Open Source License, *supra* note 48, § 3.1 (same); Sun Public License, *supra* note 48, § 3.1 (same).

The infectious terms in these licenses, however, are softened by a provision allowing for the creation of a larger work that does not have to be released under the same license terms. *See* Mozilla Public License, *supra* note 48, § 3.7 ("You may create a Larger Work by combining Covered Code with other code not governed by the terms of this License and distribute the Larger Work as a single product."); Nokia Open Source License, *supra* note 48, § 3.7 (same); Sun Public License, *supra* note 48, § 3.7 (same). A "larger work" is defined as "a work which combines Covered Code or portions thereof with code not governed by the terms of this License." Mozilla Public License, *supra* note 48, § 1.7; *see* Nokia Open Source License, *supra* note 48, § 1; Sun Public License, *supra* note 48, § 1.7. When a "larger work" is created, the terms of the license still apply to the code originally covered. Mozilla Public License, *supra* note 48, § 3.7; *see* Nokia Open Source License, *supra* note 48, § 3.7; Sun Public License, *supra* note 48, § 3.7.

not coupled and integrated with a law school merely because both are in the same university.

Second, works identifiably independent and separate, or merely aggregated, are unlikely to be derivative works of each other in a copyright sense.[146] The safe harbors amplify this point, and build it into the license permissions implemented by the GPL's open source approach. Regardless of whether an open source license implements a contract between contributing programmers and users/redistributors,[147] the license grants conditional permissions to users and redistributors. As long as one complies with the conditions, she can redistribute modified or unmodified open source software knowing that any technical violation of copyright's reproduction right or derivative work right is excused. In the GPL, the safe harbors draw a relatively bright line that is perhaps redundant (from a non-contract perspective) because it is likely beyond anything captured by the derivative work rubric. While it is unlikely an identifiably independent and separate program, or a merely aggregated program, would be a derivative work, the safe harbor states that permission is granted to package the open source software with other software in this way. It says one can do something they would likely be able to do anyway under copyright law.[148]

[146] *See* Loren, *supra* note 32, at 76-90. Some copyright doctrine in a non-software setting challenges the notion that mere juxtaposition would not create a derivative work. Mirage Editions, Inc. v. Albuquerque A.R.T. Co., 856 F.2d 1341, 1342-44 (9th Cir. 1988) (holding that the defendant prepared a derivative work by removing individual images from an art book and mounting them on ceramic tiles). *But see* Lee v. A.R.T. Co., 125 F.3d 580, 581-82 (7th Cir. 1997) (finding, under the same facts, images mounted on ceramic tiles did not constitute a violation of the derivative work right, and noting that scholarly criticism of *Mirage* has been pervasive).

[147] *See* McGowan, *Legal Implications*, *supra* note 4, at 289-302 (describing and analyzing a number of the potential doctrinal questions of contract law raised by the GPL); Stephen M. McJohn, *The Paradoxes of Free Software*, 9 GEO. MASON L. REV. 25, 62-63 (2000) (discussing the enforceability of open source licenses, noting that the GPL's model of assent by use is a low threshold "going beyond even conventional clickwrap and shrinkwrap provisions"); Daniel B. Ravicher, *Facilitating Collaborative Software Development: The Enforceability of Mass-Market Public Software Licenses*, 5 VA. J.L. & TECH. 11, Part III.A, ¶¶ 43-52 (2000), *at* http://www.vjolt.net/vol5/issue3/v5i3a11-Ravicher.html (discussing leading cases that might bear on the enforceability of mass-market public software licenses).

[148] *See* Loren, *supra* note 32, at 76-90. Loren argues on policy and doctrinal grounds that the "derivative work right should not extend to encompass integrated works which do not copy any of the portion of the works they may reference." *Id.* at 76.

There is a slight difference between the situations analyzed by Loren and dynamic linking of executable software in the runtime environment. Dynamic linking may copy some

Because the GPL's safe harbors draw a clearer boundary, they are more useful than its delineation of the "whole" to define the ambit of its infectious terms. On its face, the GPL gives little hint about what is included in the "whole." However, one reasonable (and oft assumed) interpretation is that the "whole" incorporates copyright's conception of a derivative work.[149] The GPL refers to works included in the "whole" as a "work based on the Program" which "contains or is derived from the Program or any part thereof."[150] This suggests that the GPL intends to apply the copyright derivative work standard to delineate whether other software is included in the "whole." Certainly, some commentators have read the GPL this way.[151]

There are several consequences of defining the GPL's infectious "whole" by reference to copyright's derivative work, including uncertainty and the emergence of alternative guidelines not specified in the license's text. The uncertainty is inherent in the derivative work rubric as applied to modified, intermixed, and coupled software.[152] Some of these alternative guides come from practice among open source programmers, and one in particular comes by way of comparing the GPL to a little-cousin license also promulgated by the Free Software Foundation: the Lesser General Public License (Lesser GPL).[153] I take up each of these alternatives in turn.

information from the invoked code into the calling program's process. The programmer of the invoked code determines what information is transferred. In essence, the transferred information is a specification of (or access point to) what is available for use in the invoked code. Even with such copying, however, it is possible that dynamic linking will fall under Loren's analysis if the information copied: (i) is unprotectable functionality; (ii) is public domain data/protocol specifications or information; or (iii) is otherwise not copyright protectable expression. Furthermore, such copying is unlikely to meet either a "substantial similarity" or "substantial incorporation" standard for limiting the reach of the derivative work right. *See id.* at 85 (describing these two standards as fundamental to limiting the derivative work right).

[149] *See* Gomulkiewicz, *supra* note 23, at 89-92 (discussing several different possible interpretations of section 2(b) of the GPL and questioning whether it "treats 'derived works' as something different and potentially broader than copyright derivative works"); McGowan, *Legal Aspects*, *supra* note 7, at 17 (noting that "any program that qualifies under copyright standards as a work derived from a GPL'd work must itself be released under the GPL").

[150] GPL, *supra* note 8, §§ 2, 2(b).

[151] *See* sources cited *supra* note 149.

[152] *See* sources cited *supra* notes 32-34.

[153] GNU Operating System - Free Software Foundation, GNU Lesser General Public License, *at* http://www.gnu.org/copyleft/lesser.html (last visited Dec. 20, 2004) [hereinafter Lesser GPL].

At one point, the FSF used the name Library General Public License for the Lesser GPL. *Id.* The original label associated the license with shared libraries – executable object code

A. User Programs and the Linux Kernel

Among practices that specify the GPL's infectious reach in various contexts, the most important one appears in the heart of GNU/Linux. Early in the Linux kernel project its originator, Linus Torvalds,[154] determined that user programs[155] licensed under terms other than the GPL, including proprietary terms, may make normal system calls to the otherwise

(typically in separate container files) available, or invoke-able, during the runtime environment for other software programs to reference and execute.

The FSF renamed the license as part of a general effort to deemphasize it. *See* Richard Stallman, *Why You Shouldn't Use the Library GPL for Your Next Library*, *at* http://www.gnu.org/licenses/why-not-lgpl.html (last visited Dec. 20, 2004) [hereinafter Why-Not-LGPL] (describing that, in the future, the GNU project will apply the GPL rather than the Lesser GPL to many software libraries, and discussing some of the relative advantages and disadvantages of each, arguing that "[u]sing the ordinary GPL for a library gives free software developers an advantage over proprietary developers: a library that they can use, while proprietary developers cannot use it").

[154] *Contributors*, *in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION, *supra* note 2, at 269-70 (describing Linus Torvalds as the creator of the Linux kernel).

[155] Torvalds' use of the term "user programs" requires explanation of another technical concept, albeit in a generalized way ignoring many technical nuances. Recall that the computer runs many processes, each of which is a program in execution, i.e., a program running under the supervision of the operating system. Also recall that the kernel of the operating system can be thought of as the first and most powerful process among all the processes running in the computer. *See supra* text accompanying note 139; *infra* note 154. Due to the superiority of the kernel "process," it is said to run in "kernel mode." For present purposes, take this to mean that only the kernel can interact directly with the computer's physical hardware. User programs cannot interact with the hardware. Thus, they run in "user" mode during normal execution. User programs, however, when they make normal system calls, are using/running the kernel's code, and thus may momentarily switch to kernel mode for these operations.

Given this technical explanation, part of Torvalds' confidence that user programs making normal system calls are not infected by the GPL comes from this important technical distinction. The technical distinction prompted his view that normal system calls are not works derived from the kernel. In the class of UNIX-like operating systems to which GNU/Linux belongs, the technology makes this fundamental distinction between kernel mode and user mode. The distinction supports Torvalds' intuition that the user processes are not derived works merely by making normal system calls. Regardless, Torvalds followed the intuition with a written statement when he originally put the Linux kernel under the GPL. Torvalds, *The Linux Edge*, *supra* note 2, at 109. That statement can reasonably be taken as a modification to the GPL's broad infectious terms. In effect, Torvalds modified the GPL, and licensed the kernel under the modified GPL.

GPL-licensed Linux kernel.[156] Torvalds describes this determination as one of the most important non-technical decisions enabling the Linux kernel's success.[157] His decision facilitated compatibility by removing a degree of uncertainty about the GPL's infectious terms.

Within a specific, yet important technological context, concerns about what would be included in the "whole" were diminished. As a result, vendors of proprietary software packages were willing to offer versions of their products that ran on the Linux kernel, knowing Torvalds' decision lowered the risk of claims that by merely running on the kernel their code must now be open-sourced.[158] One key example is the database product, Oracle.[159] It is the most ubiquitous commercial database software package in computing. Its availability on GNU/Linux heightens the desirability and demand for GNU/Linux-based systems. Other examples abound.[160]

[156] Torvalds, *The Linux Edge*, *supra* note 2, at 108-09. Torvalds describes his normal system call decision as follows:

> We ended up deciding (or maybe I ended up decreeing) that system calls would not be considered to be linking against the kernel. That is, any program running on top of Linux would not be considered covered by the GPL. This decision was made very early on and I even added a special read-me file . . . to make sure everyone knew about it. Because of this commercial vendors can write programs for Linux without having to worry about the GPL.

Id.

The special read-me file appears as a preamble to the GPL as applied to the Linux kernel. It specifies that rights arising from the kernel are disclaimed to exclude derivative work rights in application programs that use kernel services by normal system calls. Thus, Torvalds declared that this intermixing of the software would not be part of the infectious "whole." Torvalds, preamble to the GPL (June 1991), *reprinted in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION, *supra* note 2, app. B, at 263.

[157] *See* Torvalds, *The Linux Edge*, *supra* note 2, at 108.

[158] *See* Jerry Epplin, *Using GPL Software in Embedded Applications*, *at* http://www.linuxdevices.com/articles/AT9161119242.html (last visited Dec. 20, 2004) ("This explains why so many large software companies, such as Oracle, have released Linux versions of their proprietary products. Virtually no one contends that merely running a program under Linux obligates you to release your software under the GPL. And this conclusion is reinforced by Linus' GPL clarification.").

[159] Oracle is #1 on Linux, *at* http://www.oracle.com/ip/deploy/database/theme_pages/index.html?linux_02032003.html (last visited June 1, 2003, but later found to be unavailable at this address) (on file with author) (describing Oracle's commitment to, and product offerings for, GNU/Linux).

[160] Other prominent proprietary software companies now selling products designed to run on Linux include SAP and VERITAS. *See* SAP, SAP on Linux, Frequently Asked Questions, *at* http://www.sap.com/solutions/netweaver/linux/faq/tech_faq.asp (last visited Dec. 20, 2004) (noting that "SAP was the first software [company] in the world that run [sic]

Torvalds' practical decision implemented a practice that reduced uncertainty. The result was greater market penetration by GNU/Linux due to the greater compatibility it offered, in addition to its other benefits. The compatibility made GNU/Linux a more attractive technology platform because users could do more with it. More applications were available, increasing for GNU/Linux users the overall benefit and usefulness of a GNU/Linux-based computer system. Given these beneficial effects, it is important to understand the role of Torvalds' decision in the framework of extending, intermixing, and coupling software.

Among the three categories in the framework, (i) modify or extend software, (ii) intermingle or intermix software, and (iii) couple or integrate software, Torvalds' decision relates to the latter two categories. When a user program makes normal system calls to the Linux kernel, it is in essence intermixing itself with the kernel in the way described in the framework. Alternatively, it might be thought to be coupled with the kernel, depending on the technical implementation. The programmer has planned for the program to perform some function, but has not supplied the source code for that function. Instead, by design, the program looks to other executable object code to perform that function.

This other code is found in the Linux kernel, accessed by the "normal system call" discussed by Torvalds.[161] "Normal" Linux kernel system calls

mission-critical ERP operations on Linux"); VERITAS, VERITAS for Linux, *at* http://www.veritas.com/van/technologyzone/linux.jsp (last visited Dec. 20, 2004) ("VERITAS Software's market-proven suite of storage management, enterprise data protection and application availability technologies combine to bring greater enterprise capabilities to Linux . . . .").

[161] By necessity, the description in this paragraph generalizes a great many technical details. One detail worth mentioning is that the executable object code in the Linux kernel is itself a source of controversy over the scope of the GPL's infectious terms. In the technological framework I have developed thus far, the best way to conceive of the kernel is that it is the first and most powerful process among all the processes running in the computer. It is first because when the computer powers up and "comes to life" it runs the executable object code designated as the kernel. It is the most powerful process because it has control over the physical hardware, and remains the master scheduler of the other processes (programs in execution). It mediates all other processes' use of the hardware. They must ask for access, and ask the kernel to cause the hardware to perform hardware-related work for them. The kernel decides when to give them computing time for their work, and when they have to wait for other processes.

The Linux kernel is modular. Programmers other than Torvalds and other kernel programmers can use a standard approach provided by the kernel to add "modules" to the kernel. In my framework, this falls into the middle category, intermingling or intermixing are best defined for our purposes as those that are documented as such (i.e., the interface specification for the system call is published).[162] Whether the program's use of a normal system call is in either the second or third category is less important than realizing why it is not in the first: the program's source code and the Linux kernel's source code are not melded together in the development environment to execute under common control in the runtime environment. The Linux kernel is not extended or modified by bringing into it the program's source code. The two sets of source code remain separate in the development environment, classifying their interactions into the second or third category.

Torvalds clearly meant to draw a boundary in an important technological way in order to reduce the uncertainty associated with the GPL.[163] He staked his boundary in a special "read-me" file. Open source software usually carries its license along with the files holding the software. The open source license in use, such as the GPL, may be stated in a file in its entirety. In addition, each of the source code files may contain delineated,

code in the runtime environment, although it is different in certain ways because the other programmers are intermixing their code with the operating system kernel. The Linux community technical term for this capability is "loadable kernel modules," or LKM. Use of LKMs has been a source of GPL controversy because some vendors have provided LKMs that are not under the GPL. One common use for a LKM is to provide a device driver. A device driver helps the Linux kernel communicate with a hardware device.

For example, assume: (i) that a vendor provides a LKM device driver for the kernel to interface with a new graphics card; (ii) that the graphics card is popular and many GNU/Linux users want to use it; and (iii) that the vendor is unable to disclose the source code of the device driver, a restriction which could occur for many reasons. Since the GPL covers the Linux kernel, the question is whether its infectious terms should cause the LKM device driver to go under the GPL. If so, the device driver will not be available for Linux, which could very well mean that the desired graphics card might not be available for use with Linux (typically, hardware manufacturers are best positioned to write or commission device drivers for their hardware). In this example, the threat of GPL infection eliminates the availability of the graphics card. The example illustrates the general effect on LKMs arising from a GPL protected Linux kernel. LKMs are a useful technological approach to extend the kernel or increase the entire operating system's interoperability with the greater world. These benefits engender the controversy because some LKM device drivers are not GPL protected.

[162] *See* Email from Linus Torvalds to a redacted questioner (Oct. 19, 2001, 13:16:45 PDT) (discussing public, published software interfaces for the Linux kernel and noting that when an "interface is published, [it is meant] for external and independent users. It's an interface that we go to great lengths to preserve as well as we can, and it's an interface that is designed to be independent of kernel versions."), *at* http://www.atnf.csiro.au/people/rgooch/linux/docs/licensing.txt (last visited Dec. 20, 2004) [hereinafter Torvalds Email].

[163] *See* Torvalds, *The Linux Edge*, *supra* note 2, at 108-09.

license-identifying information. For example, a source code file might have a “comment” statement[164] near its beginning, saying that it is licensed under the GPL.[165] While a variety of methods might work to communicate the licensing information, Torvalds’ approach was effective to provide notice that his view of the GPL, as applied to the Linux kernel, would allow non-GPL-protected software to run on an operating system based on the Linux kernel. Another way to look at this, which I prefer, is to say that the Linux kernel uses a slight variation of the GPL license. I call this variation the “GPL with immunization for normal calls,” or GPL-INC.[166]

[164] A comment statement is non-functional. The compiler ignores it when it translates the source code into the executable object code. Most source code typically contains comments, and they are used in the ordinary sense of the word – to comment on the code. Even though not used in the object code, comments can play an important role in making the source code valuable. Some programmers document the operation of the code through comments. These efforts are sometimes extensive, and sometimes minimal. The need for comments to explain the code may depend on the complexity of the problem being solved and the programmer’s skills. A general axiom is that a sprinkling of comments can help the readability and understandability of a well-designed and well-written program, particularly at junctures in the logic. But, continuing the axiom, commenting, no matter how extensive, is unlikely to make a poorly designed program with unnecessarily complex and hard to understand structure readily understandable. One can often glean the history associated with a program’s development from the comments, including approaches tried and abandoned, particularly troubling problems encountered and solved, and perhaps a record of who programmed particular sections of the source code. Finally, as mentioned in the text, sometimes even legal notices, such as a copyright or licensing notice, are placed in source code comments.

[165] Another approach is that used by Torvalds: put licensing information into a file named according to a convention (“read-me”) that signals one to read the file’s contents.

[166] If one interprets Torvalds’ “immunization for normal system calls” notice as a license permission, then one characterization is that Torvalds wrote his own license. He did so by copying the GPL, but then further specified a technical scenario where the GPL’s infectious terms did not apply, resulting in the GPL-INC (GPL with immunization for normal calls). In this scenario, of course, Torvalds’ license is mostly the original work of others, but his set of permissions is different by way of greater specificity. Assume for a moment that Torvalds did this with his first version of the Linux kernel code. As later programmers contributed code, establishing the web of licensing interdependency, are they bound by the GPL or the GPL-INC? *See* Torvalds Email, *supra* note 162 (responding to a questioner’s query whether the other programmers who contributed to the Linux kernel agreed to using a modified GPL (i.e., a GPL with an exception to its infectious terms), by stating that “[t]he ‘user program’ exception is not an exception at all, for example, it’s just a more clearly stated limitation on the ‘derived work’ issue. If you use standard . . . system calls . . . your program obviously doesn’t ‘derive’ from the kernel itself.”). In my hypothetical, the contributing programmers would simply have developed the Linux kernel under the GPL-INC, without, of

Perhaps Torvalds simply believed that user programs were not derivative works of the kernel - that they are identifiably independent and separate software.[167] Torvalds understood the GPL's infectious terms in relation to the copyright derivative work standard.[168] Because they interact, user programs running on a Linux-kernel-based operating system seem to be more than a mere aggregation of the two. Thus, the first GPL safe harbor would not apply, but the second might. The theory would be that user programs are not derivative works (i.e., are not part of the GPL's "whole" when they use published, normal system calls). In this sense, they remain identifiably independent and separate because they employ only the published functionality given by the Linux kernel for all programs to use.

Neither the GPL's safe harbors, nor its concept of infecting the "whole," provided Torvalds sufficient certainty. This caused him to specify that a stated practice would not trigger the GPL's infectious terms, at least as applied to the Linux kernel. This practice provided an alternate guide, providing certainty where the GPL's open-ended concept of the "whole" gave less certainty.

The next section looks at a second alternative guide; comparing the GPL to a little-cousin license also promulgated by the Free Software Foundation: the Lesser GPL.[169]

course, ever using that name for the license conditions. Regardless of nomenclature, they knew about the Torvalds' exception for user programs making normal system calls.

[167] *See id*. (describing Torvalds' view that user programs making normal system calls are not derived works). *But see* Torvalds, *The Linux Edge*, *supra* note 2, at 108-09 (questioning whether the GPL's derived work standard for the "whole" would encompass user programs making system calls, and noting that, due to the potential ambiguity, they were not).

[168] *See* Torvalds, *The Linux Edge*, *supra* note 2, at 108 (describing the GPL's infectious terms by reference to the copyright derivative work standard); Torvalds Email, *supra* note 162 (describing Torvalds' view that the GPL's infectious terms relate to the copyright derivative work standard).

[169] *See supra* note 153 (introducing the Lesser GPL). "[T]he GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License." Lesser GPL, *supra* note 153, preamble.

B. The Lesser GPL – Intermixing Libraries and Other Software

In contrast to the GPL's open-ended reference to the "whole" to suggest the reach of its infectious terms, the Lesser GPL is more specific. Its terms are infectious in ways narrower, but similar to, the GPL.[170] The Lesser GPL, however, adds an additional safe harbor: use of the software as a library does not trigger infectious terms. The library itself must be distributed as open source software, but the calling software need not be. Within my framework, use "as a library" means that the Lesser GPL protected software is intermixed with other software in either the development or runtime environment. The programmer has either planned that the other software relies on code from the library at runtime or statically linked her code to the library's code. The Lesser GPL safe harbor allows specific forms of intermixing,[171] but in doing so provides a wealth of commentary about its

[170] The Lesser GPL is structured following the GPL's outline. Many of the terms not of concern to my inquiry are the same or similar, such as provisions concerning warranties and avoiding patent rights. The Lesser GPL's infectious terms also involve copyright's derivative work standard for its rubric of the "whole." It includes its own mere aggregation safe harbor. It also includes an "identifiably independent and separate" work safe harbor. Beyond these features that are similar to the GPL, however, the Lesser GPL describes several additional safe harbors for using the software as a library with proprietary software.

[171] The Lesser GPL specifies several rather technical safe harbors from its infectious terms. The technical complexity relates to the previously presented framework, in particular to ways of modifying, extending, and intermixing code. Section 5 describes in detail what combinations are considered derivative works that fall under section 6. *See* Lesser GPL, *supra* note 153, §§ 5, 6. The license grant of section 6 allows one to distribute other software under terms of her choice (to a degree), but the library covered by the Lesser GPL must remain open source software:

> [Y]ou may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.
>
> You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. . . .
>
> . . . .
>
> It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

Id. § 6. There are additional requirements to preserve the open source status of the original library, and to require certain technical provisions for use of a shared library. *Id.* Also, the Lesser GPL includes another safe harbor for distributing new libraries based on the original big-cousin license, the GPL. In addition, like the practice promulgated by Torvalds for the Linux kernel, the Lesser GPL is an alternative to the GPL's rubric of the "whole" for infectious license terms. Of course, to use this alternative, a programmer would have to initially license her project under the Lesser GPL.

The FSF developed the Lesser GPL along with the GPL, but later deemphasized the Lesser GPL.[172] It did so believing that the Lesser GPL did not promote the interests of open source programmers as well as the GPL. If programmers put their best open source software under the GPL, proprietary developers would be unable to use it, giving open source programmers a competitive advantage.[173] The FSF also reasoned that the infectious terms of the GPL would give proprietary software an incentive to succumb to the GPL license in order to interoperate with best-of-breed open source applications. The FSF reports at least one such occurrence of this.[174] The incentive to succumb supposes an aggregate effect: as more very good open source software is available under the GPL, it creates a superior platform for software development, leading to an upward spiral of superior software and

library (the new library must remain open source software) along with other libraries not covered by the Lesser GPL. *Id.* § 7.

[172] *See* GNU Operating System - Free Software Foundation, GNU Library General Public License Version 2 (June 1991), *at* http://www.gnu.org/copyleft/lgpl.html (last visited Dec. 20, 2004) [hereinafter Library GPL] (stating "that the GNU Library General Public License has been succeeded by the GNU Lesser General Public License"). In its preamble, the Lesser GPL suggests that the programmer consider using the GPL rather than the Lesser GPL: "think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case." *See* Lesser GPL, *supra* note 153, preamble.

[173] *See* Why-Not-LGPL, *supra* note 153 ("Proprietary software developers have the advantage of money; free software developers need to make advantages for each other. Using the ordinary GPL for a library gives free software developers an advantage over proprietary developers: a library that they can use, while proprietary developers cannot use it.").

[174] *See id.* The "Why-Not-LGPL" document describes in full the attraction it envisions for infectious open source software.

> [W]hen a library provides a significant unique capability, like GNU Readline, that's a horse of a different color. The Readline library implements input editing and history for interactive programs, and that's a facility not generally available elsewhere. Releasing it under the GPL and limiting its use to free programs gives our community a real boost. At least one application program is free software today specifically because that was necessary for using Readline.

Id.

increasing open source use.[175] On the other hand, the Lesser GPL acknowledges that in some cases the less infectious license is the better choice.[176]

One such case is the application of the Lesser GPL to a key library used in GNU/Linux. Recall that Torvalds' designated that user programs making

[175] *See id.* The "Why-Not-LGPL" document describes the aggregating effect as follows:

> If we amass a collection of powerful GPL-covered libraries that have no parallel available to proprietary software, they will provide a range of useful modules to serve as building blocks in new free programs. This will be a significant advantage for further free software development, and some projects will decide to make software free in order to use these libraries. University projects can easily be influenced; nowadays, as companies begin to consider making software free, even some commercial projects can be influenced in this way.
>
> Proprietary software developers, seeking to deny the free competition an important advantage, will try to convince authors not to contribute libraries to the GPL-covered collection. For example, they may appeal to the ego, promising "more users for this library" if we let them use the code in proprietary software products. Popularity is tempting, and it is easy for a library developer to rationalize the idea that boosting the popularity of that one library is what the community needs above all.
>
> But we should not listen to these temptations, because we can achieve much more if we stand together. We free software developers should support one another. By releasing libraries that are limited to free software only, we can help each other's free software packages outdo the proprietary alternatives. The whole free software movement will have more popularity, because free software as a whole will stack up better against the competition.
>
> Since the name "Library GPL" conveys the wrong idea about this question, we are planning to change the name to "Lesser GPL."

Id. (noting that the name was in fact later changed from Library GPL to Lesser GPL).

[176] *See* Lesser GPL, *supra* note 153, preamble. The reasons that the FSF sees as legitimate for using the Lesser GPL include the following:

> For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.
>
> In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Id.

normal system calls to the kernel did not invoke the rubric of the GPL's infectious "whole." The FSF did something similar that increased the compatibility of GNU/Linux with proprietary software. It released library software under the Lesser GPL, acknowledging that doing so "enables many more people to use . . . the GNU/Linux operating system."[177] The library provides a set of standard functions available in a popular programming language. Thus, by cutting back on the GPL's infectious terms, the Lesser GPL allows programmers to combine open source software organized in a specialized way (as a library) with non-open-source code. This acknowledges the value in compatibility and interoperability, particularly for ubiquitous or standardized interfaces and interchange capabilities.

Besides its significance as a license in its own right, the Lesser GPL is important for its commentary about the GPL. Given the lack of specificity about what falls into the GPL's infectious "whole," any statements by its authors on the matter are valuable. Perhaps surprisingly, the Lesser GPL, quoted below, is more specific about what falls into the "whole" than the GPL.

> 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. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.[178]

Thus, in terms of my framework, the Lesser GPL says that the GPL's infectious "whole" is intended to capture software intermixed in both the development and runtime environment.[179]

[177] *Id.*

[178] *Id.*

[179] Interestingly, the earlier version, the Library GPL, put the derivative works issue in slightly different terms:

> The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.
>
> Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers

By reflecting on the GPL, the Lesser GPL provides a second example to understand the envisioned scope for the GPL's infectious terms. The Lesser GPL states that the GPL considers both static and dynamic links as derivative works of the GPL protected software. In that case, the rubric of the "whole" requires the other software to succumb to the GPL terms. If distributed in an intermixed form, the other software must go forward in that combination as open source software.

This example, along with the earlier example recounting Torvalds' decision about normal system calls, adds niche certainty to the otherwise indeterminate rubric established by the GPL. To see this contribution against the framework developed in Part III, the table below maps the two examples discussed in this part against that framework.

> did not use the libraries. *We concluded that weaker conditions might promote sharing better*.

See Library GPL, *supra* note 172, preamble (emphasis added).

Table 2 – Certainty Enhancing Measures within the Infectious Framework

Category	Modify or Extend	Intermingle or Intermix	Couple or Integrate	Mere Aggregation
Description	Software in a single container or file, modified; or extended by copying code from external sources	Software in multiple containers, intended to function under the control of a single process, linked at development or runtime	Software in multiple processes, interacting and interdependent to some degree	Separate software on the same media
GPL	Rubric of the "whole"			
Torvalds practice under the GPL	Likely infected by the "whole"	User programs not infected by the "whole" But, for loadable kernel modules, it depends on technical details[180]	Likely not infected by the "whole"	Not infected by the "whole"
Lesser GPL	Likely not infected by the "whole"	Not infected by the "whole"	same	same

This mapping shows only a few examples. There are many more technological options to modify, extend, intermix, or couple software than the licenses and practices thus far discussed. This leaves uncertainty about this continuum and the GPL's infectious terms for two primary reasons. First, the scope of the derivative work right in copyright is underdeveloped along this continuum. Besides its notorious status as the most uncertain of copyright's entitlements, case law applicable to this technological framework is sparse, or outdated by the pace of advancing technology.[181] Second, even

[180] The issues arising from intermixing loadable kernel modules, or LKMs, with or within the GPL protected Linux kernel are discussed *supra* note 161. Some LKMs are distributed under a proprietary license, without the source code. Thus, their intermixing with the Linux kernel may violate the GPL's infectious terms.

[181] Advancing software development techniques that outpace case law on copyright's derivative work right include programming with "objects." This is also called object-oriented programming. Certain modern programming languages are "object oriented" because the language models the real world using software "objects." The key point for purposes of this

with the examples discussed in Part IV, the GPL leaves many questions unanswered about how far it might reach along this framework.

The next part analyzes these two sources of uncertainty for incentives in open source software development. The analysis questions the efficacy of the GPL's infectious terms, or any set of infectious terms so broadly specified, to systemically harvest existing and future-developed software into the open source fold, and to support the other key tenets of open source software: collaborative community development, anti-privatization, and source code availability. It is not my intent to pass judgment on the larger goal, promoting development of open source software, but rather to question whether an infectious implementation will achieve such a goal given that infectious uncertainty chills other important objectives, including interoperability and compatibility.

V. THE INCENTIVES EFFICACY OF INFECTIOUS LICENSE TERMS

An infectious open source license operates along the technological framework presented in Part III, endowing its software with a dubitable zone of infectious risk for other software. This is intended to transform the other software into free and open source software and protect the key tenets of the open source approach: source code availability and royalty-free software. The GPL expresses the zone's outer boundary by its rubric of the entire "whole."[182] The power behind the zone is the threat of copyright infringement for reproduction and derivative work rights violations. The analysis below assesses the efficacy of this zone in order to determine its incentive effects on free or open source software, and on proprietary software.

Ironically, important precursors to GNU/Linux's growth quarantined the zone of infection in a few specific ways, even though most of this flagship open source operating system employs the GPL, thus disseminating a potentially wide zone of infection. First, early in the Linux kernel project, Linus Torvalds signaled that he did not consider user programs making

Article is that objects intermix and couple with the programs that use them in analogous but different ways from what is described above in the text. Thus, the increased use of object-oriented programming further increases the uncertain reach of copyright's derivative work right for software works.

[182] GPL, *supra* note 8, § 2, ¶ 1.

normal system calls to fall in the zone.[183] Second, the GNU project put important software libraries for a popular programming language under the Lesser GPL,[184] a license written by the very same organization that promulgated the GPL. The Lesser GPL allows proprietary software a degree of meaningful intermixing with the open source software under its purview. These two precursors to the impressive growth and popularity of GNU/Linux suggest a rethinking of approaches that put a broad, or uncertain, scope for infectious terms first and foremost. Considerations of interoperability and compatibility should carry equal or greater weight, especially for open source software competing in a platform space[185] where network effects resulting from compatibility are important to market penetration and growth.[186]

[183] *See supra* Part IV.A.

[184] *See supra* Part IV.B.

[185] *See* Douglas Lichtman, *Property Rights in Emerging Platform Technologies*, 29 J. LEGAL STUD. 615, 615 & n.1 (2000) (describing a "platform" as something a consumer can purchase "to enhance the value of some number of independently purchased goods," and offering operating systems as an example of a platform technology, while describing application software that runs on the platform operating system as "peripherals" to that platform).

[186] It seems that there may be two network effects in play for open source software, which could either compete or compliment, depending on the circumstances. First, using GNU/Linux as an example, it becomes more valuable as more users adopt it. Second, however, it may become more valuable as it interconnects with more systems and software. *See id.* at 616-18 (describing how platform technologies, such as operating systems, are more valuable to users as they support more peripherals, which include applications). These two effects are related, and, typically, one might expect them both to contribute to the value of a technology.

But in the context of open source software, expansive infectious terms might be understood as an attempt to play them against each other. Infectious terms inhibit interconnectivity with non-open source software while the rest of the license maximizes it for other open source software. Given the cost advantages (and purported quality advantages) of open source software (although its alternative cost is not zero – some believe it has higher operating cost than proprietary software), this could have the effect of accelerating the time to a tipping point at which GNU/Linux becomes the dominant operating system. By minimizing interconnectivity with proprietary systems and software, and offering sufficient attractions as a platform technology for any open source applications, systems, and software, the first network effect eventually swamps the second.

Whether playing the two effects against each other with infectious terms produces this result depends on a much more in-depth analysis than I can proffer here. This suggested explanation comports with some of the FSF's reasons for infectious terms: it wants to keep the benefits of high quality open source code sequestered among the open source programmers so they can create advantages for each other. *See supra* notes 172-76 and accompanying text.

To express my analysis, this part constructs a legal framework over the technological framework that measures the scope of the GPL’s infectious licensing terms. My purpose is to show, by creating an example from the GPL’s rubric of the “whole,” that several alternative legal views of the infectious scope issue reduce to approximately the same inquiry for my purposes. Most infectious licenses of concern to my analysis will have some word, or short phrase, that harkens to copyright’s derivative work right.[187] In the GPL, this term is the “whole.” The question then becomes: what is the copyright or contract/permission effect of this term when one complies with all open source license conditions except the infectious terms’ operation on the “other” proprietary software?

The first step to answer this question is to narrow the fact setting of my inquiry. To isolate and focus on infectious terms, the next section describes scenarios where programmers combine software such that the main issue of GPL compliance is the infectious terms.

A. Isolating the Infectious Terms

Two examples provide the setting to illustrate isolation of the GPL’s infectious terms.

As a first example, recall the earlier discussion of a sixty-page computer program being analogous to this Article.[188] Suppose someone extensively rewrote this Article, keeping to some degree the original structure, sequence, and organization, but leaving no paragraph untouched. If the modifications were instead to a computer program, such revisions suggest a non-literal infringement analysis for copyright’s reproduction right.[189] Now assume that the program is open source software. The GPL will allow such revisions, but they may become part of the “whole.”

If the extensive modifications in part arrange the original code for interaction with other proprietary software, then the resulting work, intermixed at development or runtime, might fall in the infectious zone. If so, however, note that the modifying developer could theoretically comply

[187] The licenses of concern have broad, uncertain language and scope. A license with a well-drafted, detailed phrase delineating infectious terms will probably be more certain, and of less concern.

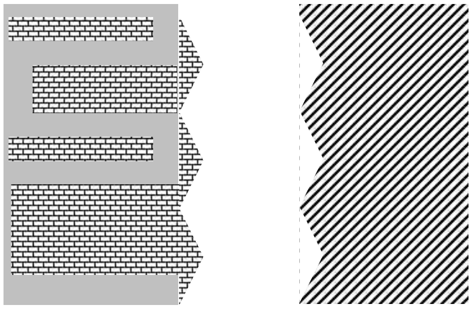
[188] *See supra* notes 129-33 and accompanying text.

[189] *See* Computer Assocs. Int’l v. Altai, Inc., 982 F.2d 693, 706-11 (2d Cir. 1992) (applying what would later become the leading test for analysis of non-literal copyright infringement of the structure, sequence, and organization of computer software).

with many, if not all, of the other open source conditions. She could provide the modified source code for the open source part of the resulting work, yet withhold source code for the proprietary part. She could charge royalties only for the proprietary part.[190] The area carrying the greatest risk of noncompliance is the infectious terms. And under some judicial analysis of the derivative work right, the modifications in this example likely run afoul of that right.[191]

Figure 4, below, styled from the earlier illustrations of intermixed software, shows this first example of a highly modified sixty-page program, with the modifications intended to fit into a proprietary program.

Figure 4 – Modify the Original Open Source to Intermix with Other Software

	Modify to intermix	Key
Development Environment		Original open source code: ▒ Modified open source code: ▦ Proprietary code: ▨

[190] The proposition that a distributor could charge royalties only for the proprietary part assumes that there is some mechanism to counter the user/customer's limited ability to disaggregate the charged price among the proprietary and open source components, and among other complimentary services such as support, warranty, and indemnification. This mechanism is market entry by other distributors if one distributor begins charging for the open source component. For a more detailed discussion of these points, see *infra* Parts V.A.1, V.A.2.

[191] *See* Dun & Bradstreet Software Servs. v. Grace Consulting, Inc., 307 F.3d 197, 209-11, 214-15 (3d Cir. 2002) (holding that defendant's use of "Copy and Call" commands to access and run plaintiff's code from defendant's program created a derivative work, and declining to follow the *Altai* case's application of the *scenes a faire* doctrine, which for computer programs includes emphasis on interoperability and external factors influencing the code – code written in response to these factors is likely not copyrightable expression); *cf.* Mirage Editions, Inc. v. Albuquerque A.R.T. Co., 856 F.2d 1341, 1342-44 (9th Cir. 1988) (holding, in a non-software context, that the defendant prepared a derivative work by removing individual images from an art book and mounting them on ceramic tiles).

A contrasting example is where the open source software is originally created to integrate with other software. Perhaps it has interfaces, access points, and data exchange mechanisms whereby it can interact with other software without revising the original code. In this case, the two might be intermixed in either the development or runtime environment. This again raises the question of whether the resulting program, when executing as a process under the operating system, falls in the infectious zone. As in the first example, one could supply the source code for the open source portion, and price the proprietary portion to reflect only its contributed value. Again, the infectious terms present the greatest risk of non-compliance. This depends, in the GPL's case, on the reach of the "whole."

The second example is styled from *Progress Software Corp. v. MySQL AB*.[192] This case involved GPL protected software. A federal district court denied MySQL's request for a preliminary injunction based on its allegations that Progress Software distributed MySQL's software in violation of the GPL.[193] MySQL, however, obtained a preliminary injunction on trademark grounds.[194] The parties agreed to collaborate to launch MySQL's database software under the GPL. The collaboration, however, went sour,[195] in part because Progress Software's subsidiary allegedly distributed MySQL's GPL covered software in a way that violated the GPL's infectious terms.[196]

[192] Progress Software Corp. v. MySQL AB, 195 F. Supp. 2d 328 (D. Mass. 2002).

[193] *Id.* at 329.

[194] *Id.*

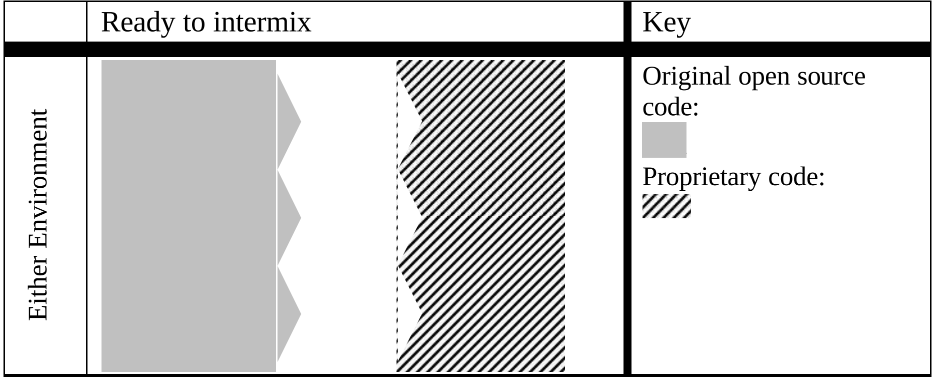
[195] *See* Henry W. Jones, III, *How a Poor Contract Sunk an Open-Source Deal*, LINUX J., Aug. 1, 2002 (noting that, for a time, many described the case as the "first litigation testing the validity and enforceability of the General Public License," attributing the parties' dispute to a poorly implemented collaboration agreement, and noting that the "judge found the GPL issue too uncertain to adjudicate in [the] litigation's early, [preliminary injunction] phase"), *available at* http://www.linuxjournal.com/article.php?sid=6025 (last visited Dec. 20, 2004).

[196] *See* Counterclaim at 5, *Progress Software Corp.* (No. 01-CV-11031) (alleging that Progress Software, through its wholly-owned subsidiary NuSphere, violated the GPL by "selling derivative works based on the MySQL Program without making the underlying source code available"); *cf.* Declaration of Eben Moglen in Support of Defendant's Motion for a Preliminary Injunction on its Counterclaims at 11, *Progress Software Corp.* (No. 01-CV-11031) (arguing that "Progress Software Corp. lost the right to distribute MySQL when it distributed NuSphere MySQL Advantage in a fashion that violated GPL") [hereinafter Declaration of Eben Moglen], *available at* http://www.gnu.org/press/mysql-affidavit.pdf (last visited Dec. 20, 2004).

The MySQL product is described as a "database engine," designed with a capability to incorporate, recognize, and use a number of "storage modules." The storage module capability allows MySQL users to choose among storage modules. This enables users to select different storage modules for different database processing applications.[197] Progress Software intermixed in the development environment the MySQL software with its own non-GPL software. It statically linked the two programs. It supplied the MySQL source code, but initially it did not supply the source for its own software. In a subsequent version it supplied all of the source code, including that for its software.[198]

Figure 5, below, styled from the earlier illustrations of intermixed software, shows the *Progress Software* inspired example of an open source program with interfaces, access points and other features designed to intermingle and intermix with other software.

Figure 5 – Open Source Software Ready to Intermix with Other Software



The *Progress Software* case is significant even though the only reported court opinion involved a preliminary injunction. First, the court was not convinced that there was a violation of the GPL's infectious terms, which it related to copyright's derivative work right.[199] Second, the issue was

[197] *See* Declaration of Eben Moglen, *supra* note 196, at 7-11 (describing the function of the MySQL software and explaining how Progress Software violated the terms of the GPL). Eben Moglen is a Professor of Law at Columbia University Law School and has served as General Counsel of the Free Software Foundation since 1994. *Id.* at 1-2.

[198] *Id.* at 9-11.

[199] *Progress Software Corp.*, 195 F. Supp. 2d at 329. The court expressed its analysis of the derivative work issue as follows, where the "Gemini program" is the Progress Software "storage module" software:

potentially moot because Progress Software supplied the source code in a subsequent version.

This Article's *Progress Software* inspired example discussed above isolates the infectious terms more than the first example of an extensively modified sixty-page program. The resulting "whole" has a proprietary software component, but the rest of the software is unmodified open source software. In *Progress Software*, the open source database engine code was designed to accommodate multiple storage modules. The case illustrates situations where infectious terms might be most disruptive to interoperability, when the open source software is built to intermix or couple with other software.

In this arrangement, there is an argument that the rest of the open source conditions are satisfied, depending on the royalty rates for use of the software: the source code for the open source part is available; users can modify and redistribute the open source part; and savvy users can still use the database engine with other storage modules.[200] The issue of license fees, however, remains.

1. Upfront Fees for Intermixed Software and Open Source Software

Assume that Progress Software charged a license fee or royalty rate for the combined database engine and proprietary storage module. This raises two issues for GPL compliance. First, the GPL allows redistributors to "charge a fee for the physical act of transferring a copy."[201] Second, it prohibits royalties of the type charged by proprietary software.[202]

> With respect to the General Public License (''GPL''), MYSQL has not demonstrated a substantial likelihood of success on the merits or irreparable harm. Affidavits submitted by the parties' experts raise a factual dispute concerning whether the Gemini program is a derivative or an independent and separate work under GPL ¶ 2. After hearing, MySQL seems to have the better argument here, but the matter is one of fair dispute. Moreover, I am not persuaded based on this record that the release of the Gemini source code in July 2001 didn't cure the breach.

Id.

[200] A savvy user who can combine the proprietary storage modules and the open source database engine raises the question of indirect liability for the distributor. *See infra* note 204.

[201] GPL, *supra* note 8, § 1. The GPL also states "you may at your option offer warranty protection in exchange for a fee." *Id.*

[202] *See, e.g.*, *id.* § 2(b) ("You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License."). *But see*

Consider two hypothetical scenarios. First, Progress Software charges an upfront fee.[203] It internally allocates that fee partly to the distribution fee allowed by the GPL. The rest is an upfront royalty for the proprietary storage module. From the purchaser's perspective, this creates ambiguity as to what she is paying for. If the proprietary storage module is part of the GPL's "whole," then the royalty is a GPL violation.

Consider an alternative to this first hypothetical. Progress Software distributes a CD containing the open source database program, with source code, and charges only for the physical transfer. On the same CD it includes its proprietary storage module, without source code, but with linkable object code. However, it does not include a linked executable file intermixing the code in the development environment. Do the GPL's safe harbors apply? Are the two works merely aggregated on the CD? Is the proprietary storage module an "identifiably independent and separate work"? Assuming that the user could readily intermix the two programs, this raises the specter of indirect liability.[204]

Gomulkiewicz, *supra* note 23, at 86-88 (hypothesizing that the GPL's anti-royalty provisions might be advisory, in which case their legal effect would be of no consequence, but noting that if the provision "is a [contractual] covenant or [copyright] license condition, a licensee must understand what fees the GPL allows").

[203] An upfront fee is the typical license approach for "off-the-shelf" software distributed through retail. The required payment is styled as a license royalty, paid up at the time of initial acquisition. This tends to make the transaction look like a purchase, raising the oft encountered issue of whether a UCC sale has occurred, or whether, instead, the contract is merely for services. *See* 2 DAVID M. EPSTEIN, ECKSTROM'S LICENSING IN FOREIGN AND DOMESTIC OPERATIONS §§ 12.50-53 (2003) (discussing the applicability of the UCC to software transactions and courts' difficulties in determining whether software should be considered a good or a service); *see also* Raymond T. Nimmer et al., *License Contracts Under Article 2 of the Uniform Commercial Code: A Proposal*, 19 RUTGERS COMPUTER & TECH. L.J. 281, 310 (1993) (noting that "custom development software contracts frequently engender litigation about whether they are contracts for goods or for services"); Andrew Rodau, *Computer Software: Does Article 2 of the Uniform Commercial Code Apply?*, 35 EMORY L.J. 853, 918-19 (1986) (discussing the drawbacks and advantages of applying the UCC to all software licensing agreements).

Many other non-retail applications in the "business-to-business" software markets also use an up-front license payment. In addition, these transactions often also include ongoing payments for support, rights to future versions, and rights to fixes in the current version of the software.

[204] Indirect copyright infringement liability comes in two forms: contributory and vicarious. *See* Douglas Lichtman & William Landes, *Indirect Liability for Copyright Infringement: An Economic Perspective*, 16 HARV. J.L. & TECH. 395, 396 (2003) (noting that although the Copyright Act of 1976 does not "explicitly recognize the possibility of indirect liability . . . courts have held third parties liable for copyright infringement under two long-

In this alternative hypothetical, it seems that Progress Software could charge a royalty without GPL risk for the proprietary module if this distribution falls into a safe harbor. What might upset this determination, however, is the possibility of indirect liability.[205] Nevertheless, this example underscores that, in some cases, programs capable of intermixing might be distributed together.

Next, assume there is an upfront charge for the CD covering: (i) a physical transfer of the open source part; and (ii) a paid-up royalty for the proprietary program merely aggregated with the open source code. This charge has the same ambiguity as the original hypothetical. The customer/user cannot disaggregate the cost factors in the price.[206] As a

standing common law doctrines: contributory infringement and vicarious liability"). For a general introduction to these two doctrines, see MARSHALL A. LEAFFER, UNDERSTANDING COPYRIGHT LAW 399–402 (3d ed. 1999).

If Progress Software included instructions or a script on the CD that showed how to link, or did link, the two sets of code, then its risk for indirect liability increases. Linking the two programs produces an executable software work that may fall under the GPL's "whole." The open source database program's source code can be linked with the proprietary object code. If a programmer/user so links the programs and redistributes the resulting program (with the source code available only for the database part), this constitutes an act of direct infringement if the infectious "whole" applies. The direct act of infringement, predicated in this example on infectious terms, raises the possibility of indirect liability running to the CD's distributor. The distributor placed components on the CD that a programmer/user might combine to create a work that violates the infectious terms, eliminating the permission that shields against copyright infringement liability. Whether indirect liability will attach depends on other factors. In light of these other factors, one would need to evaluate the elements of each type of indirect liability: contributory and vicarious infringement. *See* NIMMER, *supra* note 28, § 12.04[A][1]-[2], at 12.72 & 12.79 (noting that for vicarious liability to attach, the defendant must "possess the right and ability to supervise the infringing conduct" and have a financial interest in the exploitation of the materials, whereas contributory liability comes in "two types -- personal conduct that forms part of or furthers the infringement and contribution of machinery or goods that provide the means to infringe").

[205] *See* McGowan, *Legal Aspects*, *supra* note 7, at 24-25 (discussing the possibility that the author of a program distributed under the GPL may suffer contributory liability for copyright infringement based on the actions of a user combining the open source code with proprietary code contrary to the infectious terms).

[206] The customer's ability to disaggregate the charge will depend on her familiarity with pricing for comparable alternatives of each component. If the same open source software is available for a distribution fee from other distributors, and comparable proprietary software products are available with published prices, this might allow for price comparison of the package. This would also allow the customer to estimate the disaggregated component charges.

result, the CD distributor could in fact be charging for some of the value of the open source component under the guise of the royalty for the proprietary part.[207]

To an extent, a similar ambiguity has always existed for open source software distributors. Nothing requires them to disaggregate the fees they charge for "the physical act of transferring a copy." A distributor can charge the transfer fee in conjunction with warranty, support, and other services for a single upfront price. The more items in the mix, the greater the possibility of charging for some of the value of the open source software itself. The opportunity to do so is greatest at the point of original sale when the software media is transferred.

The primary limit to such overpricing is the relative ease of competitive entry. The main production input, the software itself, is freely available to any distributor. Thus, distributors work to differentiate themselves in a variety of ways, including traditional brand-building.[208] Differentiation and branding might allow a distributor to resist pricing pressure caused by market-entrant distributors. Of course, some open source software users will bypass distributors entirely and simply download the programs from the

[207] A number of market and economic factors will influence whether the proprietary part would bear an artificially higher price, assuming a purchaser is able to disaggregate component prices. First, if the proprietary component is in a highly differentiated product category, it will have greater resistance to pricing pressure. Second, valuation of intangibles is generally difficult and specifically obtuse for software components. This means the purchaser/user is unlikely to have precise information about the price she should pay. Third, there is a limit on the price the distributor can charge. Charging for the value of the open source part in the guise of a royalty for the proprietary component invites competition because the open source part is a cost-free input, disregarding opportunity costs. Thus, if a distributor overly inflates the overall charge, other suppliers have an incentive to produce a competitive software component that works with the open source software.

[208] For example, Red Hat, a traditional distributor of Linux products, has recently redirected its focus to the commercial arena. *See* Red Hat, Inc., Red Hat Enterprise Linux Frequently Asked Questions, *at* http://www.redhat.com/software/rhel/faq (last visited Dec. 20, 2004). Its new line of products, Red Hat Enterprise Linux (RHEL), is available under three different pricing levels, and is sold on an annual subscription basis. *Id.* Although the source code remains available under the GPL, a subscription must be purchased for every system RHEL is installed on. *Id.* The subscription includes access to RHEL binaries, upgrades, and support. *Id.* During 2004, Red Hat discontinued maintenance of its Red Hat Linux consumer line of products. Robert McMillan, *Red Hat Users Balk at Enterprise Linux Licensing*, *at* http://www.computerweekly.com/Article126419.htm (Nov. 11, 2003). Some Red Hat consumers are unhappy with the changes and claim that the new subscription agreements are "in conflict with the spirit of Linux's GNU General Public License." *Id.*

internet.[209] However, the distributor market is an important aspect of the delivery system for open source software. This is particularly true for corporate adapters, who seek to reduce the risk perceived with open source software.[210]

This subsection has reviewed upfront fees. The next subsection reviews the contrasting situation: proprietary royalties payable on an ongoing basis.

2. Ongoing Service Fees for Intermixed Software and Open Source Software

Proprietary software vendors often license their products with an ongoing royalty structure that charges for use over time. Sometimes other services are included in the regular charges, such as support, rights to new versions or fixes, distribution of new versions of the software, and documentation. These ongoing charges are characteristic of "enterprise" software.[211] These applications are unlikely to appear in a box at the local computer store. They are typically licensed in negotiated transactions and may involve tens to hundreds of thousands of dollars in software license fees, both upfront and ongoing.

Ongoing fees are harder to hide as royalty charges. Thus, a distributor who intermixes open source and proprietary software, yet who charges ongoing royalty fees, is more visible. The ongoing fees, if identified as license fees for use, will violate the open source license if the intermixed proprietary software falls into the infectious terms of the open source software. In the hypothetical based on *Progress Software*, if Progress Software charges annual licensing fees, it would violate the GPL,

[209] *See* FINK, *supra* note 6, at 4 (explaining that Linux may be purchased from a distributor or downloaded from the internet for free, and displaying a third-party study showing that approximately one-third to one-half of the Linux copies in use were downloaded for free).

[210] *See infra* Part V.C.2 (discussing the risks associated with infectious terms).

[211] Enterprise software is used in large organizations for critical operations. Thus, for example, to a national bank, its enterprise software includes the applications that keep account balances current for its millions of customers. *See* Microsoft Corp., Help and Support Glossary, *at* http://support.microsoft.com/default.aspx?scid=%2fsupport%2fglossary%2fE.asp (last visited Dec. 20, 2004) (defining "enterprise computing" as follows: "In a large enterprise such as a corporation, the use of computers in a network or series of interconnected networks that generally encompass a variety of different platforms, operating systems, protocols, and network architectures.").

unless the combined program, intermixed in the development environment, did not fall into the GPL's infectious "whole."

As in the hypothetical with upfront fees, Progress Software could distribute each component separately, perhaps on separate media or as a "mere aggregation" on the same media. This again raises the possibility of indirect copyright infringement liability. However, there must be a direct act of infringement. Such may not occur if the user only intermixes the two components and merely uses the result without distributing it. The GPL's main open source conditions primarily attach upon redistribution of modified or verbatim open source software. A user who only receives, links, and uses the two programs as a resulting work will likely not violate the GPL's conditions. Distribution or publication would, but mere use in a confined environment does not seem to be a literal problem, even if it is perhaps against the spirit and intent of the GPL.

This section began by discussing two examples of intermixed open source code: (i) modify to intermix[212] and (ii) ready to intermix.[213] The first example described an extensively modified sixty-page program. The second example described separate programs, at least one of which was originally programmed to interoperate with other code, and both of which were eventually fitted together.

Both examples, "modify to intermix" and "ready to intermix," illustrate scenarios where the infectious terms are isolated. When proprietary software is intermixed with open source software, but the source code is provided for the modified or unmodified open source component, the primary violation of the GPL may be that the infectious terms are not given effect. The main challenge to this contention, that the infectious terms can be isolated, is the condition that redistributors do not charge royalties for ongoing use. If such a fee is levied, it may be hard to show that there is no royalty charge. Royalties may not disaggregate among the open source and non-open source components. Comparable royalty rates for the proprietary component may be sparse, and none will exist for the open source part if it has no competing proprietary products.

A different type of disaggregation, however, is already a potential problem when a distributor deals only in open source software. Distributors may charge for ongoing support and other services, including the service of continually sending updates to the open source software. Thus, the royalty

[212] *See supra* Part V.A, Figure 4.

[213] *See supra* Part V.A, Figure 5.

rate issue is difficult to assess with or without intermixing proprietary code, but such intermixing moderately complicates the assessment. This weakens the case somewhat for isolating the infectious terms. On the other hand, the disaggregation problems that exist without intermixing show that infectious terms do not solve disaggregation issues.

To show how these considerations impact the efficacy of infectious terms, the next section narrows the legal question in order to apply the resulting issue to the continuum of ways to modify, extend, intermix and couple software. The section thereafter ascertains the effects of this application.

B. Efficacy Model

From a copyright perspective, an open source license expresses a generally applicable conditional permission, so the focus is whether the conditions are met.[214] For the GPL, this requires interpreting its rubric of the "whole." The power to impose this infectious condition extends at least as far as the derivative work right, but the condition need not reach that far. It could retreat from the derivative work standard.

From a contract perspective, an open source license may express an agreement between the licensor and licensee. As a contract, the promises relevant to my analysis are the open source conditions. For the GPL, viewed as a contract, the infectious terms amount to a promise to not intermix, couple, or integrate the software with non-GPL software.

Styling the GPL as a contract, however, may not help with the fundamental inquiry: what is the scope of the GPL's "whole," or the scope of its safe harbors? What remains, under either a copyright or contract analysis, is the need to deal with expansive language defining the scope of the infectious terms.[215] The GPL's "whole" is my example, but my inquiry

[214] McGowan, *Legal Implications*, *supra* note 4, at 257 & n.74. *But see* Sun Microsystems, Inc. v. Microsoft Corp., 188 F.3d 1115, 1122 (9th Cir. 1999) (holding that whether Sun could enjoin Microsoft based on a license agreement for Sun's Java technology depended on whether the license provisions in question were "license restrictions or separate [contractual] covenants"); Gomulkiewicz, *supra* note 23, at 87 ("One issue is whether these statements about fee charging are additional license conditions, a separate covenant, or merely advisory.").

[215] Expansive language delineating the scope of infectious terms could either explicitly or implicitly refer to copyright's derivative work standard, or express the scope of infection with insufficient linguistic specificity to provide much guidance, as does the GPL's rubric of the "whole." Any of these approaches leave a greater degree of uncertainty than my

is applicable to any open source license using a broad label that does not help to cabin such scope.

Many distinctions turn on whether the open source license's conditions are enforced or evaluated as conditions limiting permission, as contractual promises, or as both.[216] These distinctions could arise on a term-by-term basis; some provisions of the open source license might be part of the conditional permission, but others might not. The question whether the open source license is a contract or not adds to the overall legal uncertainty. However, this distinction is ancillary to my main motif: the scope of infectious terms and the uncertainty about their scope.[217] There has been some attention in the literature to whether the GPL and other open source software licenses are valid contracts.[218] The answer will often turn on facts

analysis herein finds beneficial, especially given the opportunity in the license itself to cabin infectious terms or express "safe harbors" from them for important purposes, such as interoperability. *See* McGowan, *Legal Aspects*, *supra* note 7, at 26 ("A rule that one program may form a derivative work of another by interacting with it would make it harder for developers to write interoperable programs.").

[216] *Id.* at 9-16 (analyzing a number of potential implications arising from styling the GPL as a contract).

[217] I do not mean to say that the contract/copyright question, or the other questions about the open source licensing approach, are irrelevant to legal uncertainty. Rather, my main focus is on uncertainty and issues arising from infectious terms. For example, the consequence for remedies is drastic if a term is found to be a contractual covenant versus a copyright permission. *E.g.*, *Sun Microsystems, Inc.*, 188 F.3d at 1122; *see also* Gomulkiewicz, *supra* note 23, at 87. However, despite the impact of this uncertainty, the type of uncertainty arising from broad infectious terms could arise in stylization of the GPL either as contract or copyright permission. Even if a license fails as a contract, the licensor can fall back on copyright. The enforceability of the contract may be more important for licensees who do not have copyright as a backstop, unless they are contributors and their programming is intertwined in the product. McGowan, *Legal Aspects*, *supra* note 7, at 13-14.

[218] *See* Patrick K. Bobko, *Linux and General Public Licenses: Can Copyright Keep "Open Source" Software Free?*, 28 AM. INTELL. PROP. L. ASS'N Q.J. 81, 102-03 (2000) (arguing that the GPL would create an enforceable contract under principles flowing from "shrinkwrap" license cases); McGowan, *Legal Aspects*, *supra* note 7, at 12 (noting that if developers properly follow the GPL's notice requirements, then the GPL resembles a typical form contract situation, but also noting that a "developer who released code with a reference to the GPL and a link to its terms would not comply with the GPL's notice requirement, and would run a greater risk of formation problems. . . . The link might go dead, for example.").

and circumstances external to the GPL's terms, such as whether there is any form of explicit or implied assent to the terms.[219]

Further complicating the copyright versus contract distinction is that the contract view may incorporate copyright concepts. Viewed as a contract, the GPL, or other open source licenses, may refer to copyright's derivative work standard to define the scope of the infectious promise.[220] The GPL seems to do this, associating in several places its rubric of the "whole" with copyright's derivative work standard.[221] Thus, the copyright and contract facets of the problem focus on the same place: language in a written instrument that expresses the reach of the infectious terms and the challenge of interpreting that language. Even if the instrument is not a contract, courts will likely use contract interpretation principles, as well as principles of equity, to construe the meaning.

If contract doctrine is taken out of the picture, the remaining copyright analysis has several facets. My discussion thus far assumes that the derivative work right extends the copyright holder's power to exclude beyond the reproduction right. Given the overlap between these two rights when non-literal infringement of the reproduction right is at issue, this

[219] McGowan, *Legal Aspects*, *supra* note 7, at 11-12 ("The GPL does not require a user to click through a dialogue box, of course, but there is nothing talismanic about that method. It is just one way of making sure that users have notice of license terms.").

[220] Contract language referring to copyright's derivative work standard raises the question of preemption. *See* Mark A. Lemley, *Beyond Preemption: The Law and Policy of Intellectual Property Licensing*, 87 CAL. L. REV. 111, 137-44 (1999) (describing that there are "two basic sets of copyright preemption doctrines. One is based on an express statutory provision partially preempting the field; the other is based on express and implied conflicts preemption."). The preemption possibility is another factor of uncertainty in the contract versus copyright analysis, but again, it is ancillary to my focus.

For open source licenses, preemption would be one of a number of ways to argue that a contract, or a provision thereof, is not in effect. If there is no contract in effect, there are no contractual promises to potentially override the licensee's ability to rely on fair use. *See* Bowers v. Baystate Techs., Inc., 320 F.3d 1317, 1323-27 (Fed. Cir. 2003) (holding that a license agreement was not preempted, and thus the anti-reverse-engineering clause in the contract overrode any fair use right of the licensee to reverse engineer the software). While fair use is part of the analysis, how it applies to infringement via violation of infectious terms is another difficult question. *See* McGowan, *Legal Implications*, *supra* note 4, at 287-89, 303 (discussing how fair use might impact an open source license, and arguing that "the open-source model of production is itself a fact that should be considered"). Furthermore, there is the additional uncertainty of the fair use standard itself, adding to the significant ex ante uncertainty of broad infectious terms.

[221] GPL, *supra* note 8, §§ 0, 2, 5, 10.

further clouds the discussion.[222] Beyond these uncertainties, there are doctrinal differences from geography. Different federal appellate circuits offer slightly differing standards for either right.[223]

My aim is not to resolve every point of uncertainty tagged above. While perhaps one could quibble with the importance of some of these uncertainties by themselves, my argument is that in aggregate they present uncertainty to a degree that influences open source software development by influencing the impact of licenses such as the GPL. Much of the uncertainty exists regardless of infectious terms, but such terms significantly enhance the ambiguity.[224]

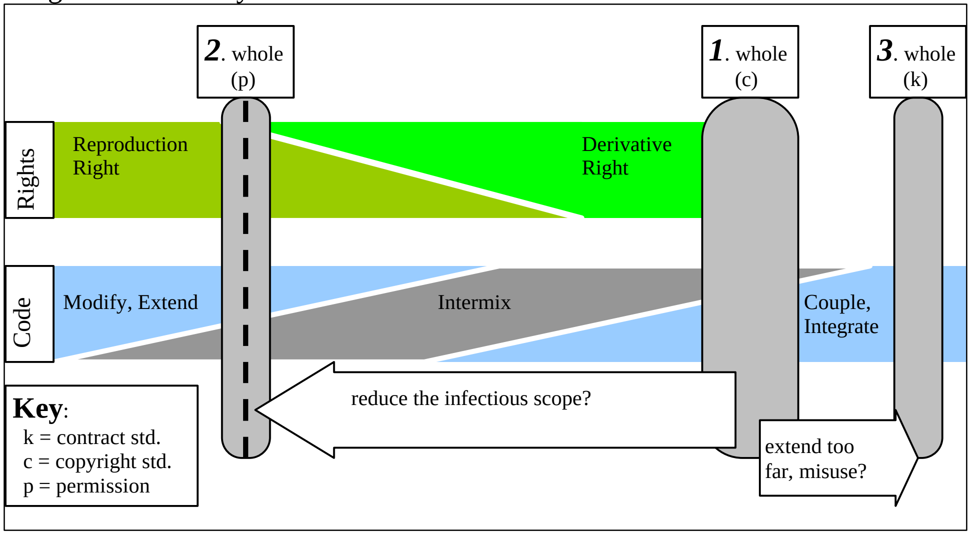
Figure 6, below, brings together these considerations. The figure is an abstraction of two overlapping continuums. First is the technological framework for infectious ambit, ranging from modifying or extending software, to intermixing it, to coupling and integrating it with other software. The second is legal; copyright's reproduction and derivative work rights. There are no hard boundaries for either continuum, so the abstraction shows overlap. Against this, the illustration shows three hypothetical possibilities for the GPL's "whole." Each scenario is numbered in the figure, with explanations below.

[222] *See* Lemley, *supra* note 108, at 1017-18; Loren, *supra* note 32, at 63-64.

[223] Dan Ravicher, *Software Derivative Work: A Circuit Dependent Determination* 4-5, *at* http://www.pbwt.com/Attorney/files/ravicher_1.pdf (Nov. 8, 2002) (discussing the disparity among the various circuits of the federal courts of appeals as to the test each has adopted to define the scope of a derivative work, and noting that several circuits "have yet to declare their definition of derivative work").

[224] *See* Zittrain, *supra* note 6, at 267-68 (noting that the copyright doctrine underlying open source software is ambiguous).

Figure 6 – Efficacy Model Illustration – scenarios for the infectious "whole"



The first scenario in Figure 6 posits that from a copyright perspective the infectious "whole" is coterminous with the outer scope of the derivative work right. It is identified by: ***1***. whole (c). In this instance, the scope and uncertainty of the infectious terms calibrate with that of the derivative work.

Although there is no empirical evidence to support the first scenario, anecdotally, this is probably the most common interpretation of the GPL's infectious "whole" - that it maps to the derivative work standard. The figure shows this standard as a gray vertical bar, drawn widely to illustrate the derivative work standard's uncertain scope.

The second and third scenarios posit that the infectious phrase in the open source license, such as the GPL's "whole," is something less, ***2***. whole (p), or something more, ***3***. whole (k), than the derivative work standard.

Scenario two, ***2***. whole (p), illustrates reducing the infectious terms by granting a permission that disclaims some scope of right potentially reachable under copyright's derivative work right. Linus Torvalds did this when he clarified that user programs making normal system calls to the Linux kernel were not infected by the GPL. In the previous discussion of this, I posited that one might even say that Torvalds applied a slight variation of the GPL to the Linux kernel, which I dubbed the GPL-INC, for "GPL with immunization for normal system calls."[225] Clearly, Torvalds' application of the GPL-INC did not move the bar to the left as far as scenario two compares to scenario one. However, there was some movement to the left along these continuums, reducing the uncertainty of infectious scope in an important technical scenario. This benefits the GNU/Linux operating system by allowing software applications, open or closed, to run on GNU/Linux, thereby increasing demand for the operating system.[226]

The vertical bar in scenario two is thinner than the bar in the first scenario. This suggests that a license standard, as opposed to copyright's derivative work, can be more precise. The licensor has the power to write a more specific standard into the permission. Although the GPL's "whole" may not create a bright line standard, a license instrument could use language that creates a specific standard. This is suggested by the vertical dotted line running through the gray vertical bar for scenario two. In many instances it would be possible to write the license permission in sufficient technical detail so as to grant a permission that gives up otherwise excludable rights under copyright.[227] The Lesser GPL is one example of this approach. It allows open source software under its purview to combine with non-open source software in a few technically defined ways. The Lesser GPL is the bright line standard that the GPL is not.

Scenario three, ***3***. whole (k), imagines an infectious standard that goes beyond the derivative work right. Such a maneuver would require enforcement of the infectious license as a contract, in order for the licensor to take advantage of the possibility to control the licensees' activity beyond the copyright power.[228] Although it seems farfetched, some commentators have alluded to the possibility of copyright misuse arising from infectious

[225] *See supra* note 166 and accompanying text.

[226] *See* Lichtman, *supra* note 185, at 617-18 (describing how demand will increase for a technology platform such as an operating system when it interoperates with more applications).

[227] *See* Gomulkiewicz, *supra* note 23, at 97-101 (arguing for an organizational process to improve and evolve open source licenses, which the author finds "buggy," and noting that such a process follows the approach of standards organizations who "create technical specifications aimed at achieving integration" of many vendors' products).

[228] *See* Bowers v. Baystate Techs., Inc., 320 F.3d 1317, 1323-27 (Fed. Cir. 2003) (holding that a license agreement was not preempted by copyright, and thus that the anti-reverse-engineering clause in the contract overrode any fair use right of the licensee to reverse engineer the software).

terms.[229] Notwithstanding this issue, there is some limit to the derivative work right. But beyond this limit, software can still couple and integrate. Thus, an open source license, enforceable as a contract, may have infectious terms reaching beyond the derivative work right. Scenario three depicts this possibility.

The efficacy model expressed by Figure 6 relates the continuum of rights with the continuum of software interaction. Language, typically in a written instrument, can grant permissions based on those rights. The permissions might be expressed by an artisan vocabulary arising from the technological continuum. Such language may have different operative impact if given effect under the law of contract versus the law that might apply to a generally applicable permission or license. In either case, the tough problem is interpreting the boundary of the infectious terms. In the GPL, this problem is

[229] Kem McClelland, *A Practical Guide to Using Open Source Software in a Time of Legal Uncertainty*, Patents, Copyrights, Trademarks, and Literary Property Course Handbook Series, Understanding Electronic Contracting 2003 The Impact of Regulation, New Laws & New Agreements, Overhead 5 (PLI Order No. G0-019Q 2003), 743 PLI/Pat 351 (listing “Potential copyright misuse?” as among the legal issues for open source software and the GPL).

Professor Nimmer notes that the defense of copyright misuse involves “extending” the scope of the copyright “monopoly” in violation of the antitrust laws. *See* NIMMER, *supra* note 28, § 13.09[A], at 13-292 (stating that there is some ambiguity as to what types of “extensions” of the copyright holder’s rights equate to copyright misuse, but noting “that such violations have been held to occur as a result of a number of copyright owners acting in combination, or alternatively, as a result of a particular copyright owner’s refusal to license certain of [its] more desirable product unless tied-in with licenses of certain of [its] less desirable product”) (footnotes omitted); *see also* Dan Burk, *Anticircumvention Misuse*, 50 UCLA L. REV. 1095, 1114-15, 1124-27 (2003) (describing how the misuse defense originated in patent law and migrated to copyright, and that “[a]lthough the importance of the misuse defense has waned in patent law, it has experienced a somewhat surprising renaissance within the law of copyright”).

Professor Nimmer puts a comprehensive discussion of misuse outside the scope of his treatise. *See* NIMMER, *supra* note 28, § 13.09[A], at 13-292. It is similarly outside my scope, except to note that its possibility, even if remote, seems greater with broad infectious terms of expansive scope. This is so because if these terms are given effect under contract in such a way as to extend the licensor’s control over the licensee beyond the reach of the derivative work right, or other rights of copyright, then the situation is analogous to copyright misuse. *See* Lasercomb Am., Inc. v. Reynolds, 911 F.2d 970, 978-79 (4th Cir. 1990) (holding that licensee’s misuse defense was successful when licensor’s agreement prohibited licensee from developing a competitive product); Burk, *supra*, at 1124-25 (discussing *Lasercomb*, and describing it as the “germinal case” to “apply misuse principles to overreaching in copyright licensing”).

particularly difficult given the minimal guidance of the word "whole," although the GPL's safe harbors offer some help.

This interpretational challenge, in light of the efficacy model, suggests two things. First, open source licenses with infectious terms should draw a brighter line for the reach of such terms.[230] Second, given the uncertain scope of the derivative work right in this area, licenses should steer clear of merely incorporating this standard to define infectious reach.[231] Both of these suggestions are further buttressed in the next section, which examines the effect of infectious terms and considers how the situation might change under these suggestions.

C. Infectious Effects

Just as there are justifications for copyright's derivative work right,[232] open source infectious terms may also have beneficial aspects. On the other hand, they may create problems, as the derivative work right can.[233] Thus, their efficacy depends on the interplay between the benefits and problems.

[230] *See* Gomulkiewicz, *supra* note 23, at 76-77. Gomulkiewicz summarizes the general need for "less buggy" open source licenses as set forth below:

> Although open source software developers may regularly fix buggy software, they do not regularly fix their licenses. There are a multitude of licenses that purport to meet the goals of open source development. These licenses reflect different, and sometimes contradictory, approaches to core licensing issues. Many of these licenses are buggy - out of date, misapplied, misunderstood and hopelessly confusing. This state of affairs benefits no one. Hackers suffer because they do not know which license form to use. End users suffer because they do not fully understand the terms of use. Commercial software developers suffer because they have difficulty discerning how open source licensed software may affect their intellectual property.

Id. (footnotes omitted).

[231] *See* Loren, *supra* note 32, at 62 ("In the context of integrated works in new technology, this broad phrasing in the statute provides little guidance in determining the appropriate scope of the derivative work right.").

[232] *See* Goldstein, *supra* note 1, at 252 (arguing that the derivative work right helps balance "the incentives required for the production of underlying works against those required for the production of derivative works," in part by generating a level of original investment in creation of new works calibrated to the expected returns from all markets, including those for the original and for the derivatives).

[233] *See* Amy B. Cohen, *When Does a Work Infringe the Derivative Works Right of a Copyright Owner?*, 17 CARDOZO ARTS & ENT. L.J. 623, 642-47 (1999) (discussing several theories by several commentators as to how and why there should be limits to the derivative work right); Lemley, *supra* note 108, at 1074 (noting that the derivative right in the copyright holder creates barriers for others who might improve upon the original creation).

The subsections below review both sides of the coin, looking at effects on both open source and proprietary software within the greater software ecosystem. On balance, I conclude that broad infectious terms of expansive scope are not desirable for open source software. On the other hand, they may be beneficial to a degree, or at least tolerable, if precisely defined and calibrated in light of other considerations.

A key assumption of my analysis is that the foreseeable future will include both types of software.[234] If true, policy concerns suggest promoting interoperability and cross-compatibility. Practically, these goals are important due to network economies in information technology. Both types of software are more valuable when they enable greater use of computing generally, and in particular the internet. Their ability to do so is enhanced by interoperability and cross-compatibility.

Copyright law also recognizes that software must interoperate. Courts have found fair use of copyrighted software when the purpose of the use is to reverse engineer the software for compatibility.[235] Recent additions to the copyright statute, granting anticircumvention rights, explicitly acknowledge the importance of allowing reverse engineering for interoperability.[236] Sometimes reverse engineering enables direct competition, but sometimes it merely facilitates interoperability. Just as there is a relationship between reverse engineering and interoperability, there is an association between infectious terms and interoperability. Thus, arguments related to reverse engineering apply, with some rework, to infectious terms. The subsections

[234] *See* FINK, *supra* note 6, at 159-67 (describing a long-term vision of coexistence between both open source and proprietary software, and positing that some of the economic characteristics of such coexistence will resemble some aspects of pioneer and generic drug company coexistence).

[235] *See* Sega Enters. Ltd. v. Accolade, Inc., 977 F.2d 1510, 1526 (9th Cir. 1992) (finding that defendant's reverse engineering of plaintiff's object code was fair use); NIMMER, *supra* note 28, § 13.05[D][4], at 13-227 (noting that the need for reverse engineering of computer software arises in the first place due to software's unique ability to hide the source code instructions in the object code, and that the reverse engineering fair use doctrine of *Sega* has held up in the paradigm case of a "user lawfully in possession of a copy of computer software in object code format to reverse engineer that code for the sole purpose of manipulating elements of that code for permissible purposes, when no other readily available substitute mechanism exists for that purpose").

[236] 17 U.S.C. § 1201(f)(1) (2000) (providing that "a person . . . may circumvent a technological measure . . . for the sole purpose of identifying and analyzing those elements of the program that are necessary to achieve interoperability of an independently created computer program with other programs").

below bring forth these and the other arguments that go to the efficacy of infectious terms.

Each of the next three subsections looks at a different aspect of the problem. The first subsection begins with the potential justifications for infectious terms. The second looks at the contrary perspective: problems caused by broad, far-reaching infectious terms. The last examines implications of these views, concluding that infectious terms need brighter lines to diminish their otherwise inefficacious impact on license certainty and software interoperability.

1. Purported Benefits of Infectious Terms

Two supposed benefits could be advanced for infectious terms. First, the FSF has argued that infectious terms reinforce open source software development and adoption.[237] Second, infectious terms may buttress the other key open source conditions: source code availability and royalty-free use.

The FSF argument presupposes competition between open source and proprietary software. It then offers reasons why infectious terms help open source in the fight. In its Lesser GPL and associated documents, the FSF argues that broad, infectious terms help prevent proprietary developers from appropriating high-quality open source software.[238] It makes the related point that some proprietary software might convert to open source in order to have these high-quality inputs, or to interoperate with them.[239] Implicit in these points is that they suppose a bandwagon effect from the incentive to succumb to open source terms: as more very good open source software is available under the GPL, more proprietary software will convert to open source, leading to an upward spiral of superior software and increasing open source use.

Undoubtedly, there is competition between open source and proprietary software. Further, empirical and anecdotal evidence suggests that open source software development has advantages for software quality, at least for some classes of application.[240] One limit to the FSF argument, however, is

[237] *See supra* notes 172-76 and accompanying text.

[238] *See supra* notes 172-76 and accompanying text.

[239] *See supra* notes 172-76 and accompanying text.

[240] Peter G. Neumann, *Robust Open-Source Software*, COMMUNICATIONS OF THE ACM, Feb. 1999, at 128; *see* Douglas C. Schmidt & Adam Porter, Leveraging Open-Source Communities to Improve the Quality & Performance of Open-Source Software 2-3 (Position paper submitted to the ACM Workshop: Making Sense of the Bazaar: First Workshop on

that the high quality of open source software is thought to systemically spring from the massive peer review process inherent in users holding source code.[241] Infectious terms that limit compatibility and interoperability may slow the adoption momentum for the software. This could lessen the number of reviewers.

Recalling the examples styled from *Progress Software*,[242] it could very well occur that more copies of the open source database engine component will be in circulation if end users can apply either open source or proprietary storage modules. More users would have the database engine source code, even if they did not have all of the source code for all of the storage modules. This could increase quality if quality derives from more "eyeballs" having access to the source to find and report bugs.[243]

Another counterpoint to the FSF's argument is the Apache web server example. This application uses an attribution-only license. It does not require the typical open source conditions and has no infectious terms, although the source code is available.[244] The Apache application continues to lead its field, suggesting that infectious terms are not necessarily a causal factor to achieve that status. Additionally, there is the intuitive inference that Apache adoption is greater in part because users do not have to worry about broad, uncertain infectious terms of expansive scope.[245] Thus, while such

Open Source Software Engineering), *at* http://opensource.ucc.ie/icse2001/schmidt.pdf (last visited Dec. 27, 2004) (describing application types, or "domains," where the authors question whether the open source approach will be successful, including niche and vertical markets, low-margin markets, and secure computing markets); *see also* Zittrain, *supra* note 6, at 282-83 (discussing the comparative reliability of open source and proprietary software).

[241] Open source's peer review debugging process is famously described by the pithy comment, "[g]iven enough eyeballs, all bugs are shallow." Raymond, *supra* note 4.

[242] *See supra* notes 192-200 and accompanying text.

[243] The counter argument to my point is that without the source code for the storage module, finding the bug in the database engine may be more difficult even when one has the source code for the database engine. Classically in software application development, one of the challenges for debugging large projects is to determine whether errors are caused by the application's program or by another program it relies on, typically the operating system. One partial solution to this problem is to require crispness, stability, and trustworthiness of the interface between the components. Open source software designed to interact with other software will likely employ standard interfaces and protocols, or well-designed access and exchange mechanisms.

[244] *See supra* notes 43, 57-59 and accompanying text.

[245] While empirical evidence is lacking as to the influence a lack of infectious terms has on Apache adoption, the GPL's infectious terms are one of its most contentious and

terms might coercively attract an occasional convert to open source licensing, the lack of such terms, or their replacement with bright-line infectious terms, may allow even greater adoption of open source software. As such, adopters would worry less about infectious terms.

For Apache, there is continued demand for the application in its entirety and for its components as packaged and offered by the open source programmers developing the Apache web server and related technology. Currently, under its attribution-only license, proprietary programmers can cannibalize some or all of Apache. The latter would be unproductive because the market demand is for the original due to its quality and other attributes arising from the open source development model. Cannibalizing pieces of Apache for proprietary products is a different concern. An open source license could prohibit this, and if so its anti-cannibalization effect might be enhanced by infectious terms.

This suggests the second potential set of benefits from infectious terms: buttressing the source code availability and anti-royalty provisions central to open source licenses. The value of source code availability might diminish somewhat if intermixed with or coupled to proprietary software without source code. Similarly, infectious terms might create a pricing buffer zone around free-use open source software. The zone helps diminish issues of price ambiguity that might otherwise exist if "free" open source software is distributed intermixed with priced proprietary software.

The source code concern is evident in the FSF's Lesser GPL.[246] That license allows open source code to intermix with proprietary code in certain specified ways. Among the conditions are several that appear to prevent the proprietary code from blockading beneficial changes to the open source part.[247] The conditions restrict the proprietary code such that even if a

discussed aspects. Apache, however, does not use the GPL. Moreover, to some extent, many see infectious terms as a risk to their internal intellectual property and seek to minimize or manage this risk when incorporating open source into their computing infrastructure. *See* FINK, *supra* note 6, at 204 (describing that information technology managers need to be prepared to evaluate the risk of mixing GPL code with other code).

[246] The source code concern is also explicitly expressed in the GPL's comments about controlling the mode of distribution for the software. GPL, *supra* note 8, § 2 (providing "it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program").

[247] Lesser GPL, *supra* note 153, §§ 6, 6(b) (describing that a proprietary program using the library of routines covered by the Lesser GPL must use: a suitable mechanism during the runtime environment to refer to those routines; suitable meaning that if a user

developer changes the open source code, it will still interoperate with the proprietary code.

Accounting for these problems is a concern for the Lesser GPL because its infectious terms are weak,[248] but for GPL software the strong infectious terms ensure that all necessary information is available for intermixing. This happens because the infectious terms force source code availability for the other work. While there are other ways for different software works to be compatible,[249] intermixing the source code in the development environment allows the greatest flexibility.

Thus, the relationship between infectious terms and source code availability is that the latter is potentially more useful when there is strong scope for the former. Given the powerful benefits that source code availability bestows on open source's collaborative development process, this suggests a policy rationale for some degree of infectious terms or for deploying the power arising from infectious terms in particular ways, such as to support collaborative development among open source software, or to support interoperability among open source and proprietary software. One limit to this argument is that most proprietary software will not succumb to infectious terms. For the foreseeable future, both types of software will occupy the market. Proprietary software, rather than shed its traditional

installs newer, interface-compatible versions of the library, the program will still work with the library; and requiring distribution of information necessary to reproduce the executable form of the combined work).

[248] *Id.* §§ 5-6 (positing a number of technical scenarios where combining a proprietary work with the Lesser GPL library might be a derivative work, but then granting certain technically-defined allowed uses "regardless of whether [the combined work] is legally a derivative work").

[249] Figure 5 in Part V.A, entitled "– Open Source Software Ready to Intermix with Other Software," illustrates software designed to interoperate. The pieces are ready to fit together in Figure 5. This allegorically shows that software can be built using components. In many cases, this is a matter of good design. The components need connections to other components. The connections are typically via standard interfaces and protocols or well-designed access and data exchange mechanisms. The classic wisdom from computer science is that with proper isolation and "layering" of the software's functionality, quality will be higher and interoperability and reusability will be greater. This relates to the source code versus object code distinction because the closed object code can have a well-documented, well-designed, and well-behaved interface. With this, one can connect open source code to the closed code to achieve interoperability. For an analysis of how intellectual property law may influence the market for software components, see generally Mark A. Lemley & David W. O'Brien, *Encouraging Software Reuse*, 49 STAN. L. REV. 255 (1997).

license upon the entreaty of infectious terms, will rebuff the overture and simply not play.[250] To the extent beneficial interoperability between the disparate software is lost, this is a cost associated with infectious terms.

Besides the potential supportive effect for source code availability, infectious terms also may support the anti-royalty provision common for open source licenses. They deter combinations like the one at issue in the *Progress Software* case.[251] This decreases situations where price disaggregation would be difficult. It is easier to determine whether a distributor is charging royalties contrary to the open source license if there are fewer components in the ultimate price the purchaser pays. Enforceability of the anti-royalty provision is thus enhanced.

While infectious terms thus may play a role in protecting the anti-royalty provision, their impact must be viewed in light of a potentially greater effect: the potential for other distributors to enter the market for the open source software. Assuming no price-fixing cartel, such competitive entry enforces the anti-royalty provision, which itself has a disaggregating effect by driving the prohibited royalty charge to near zero. Customers will be able to compare prices and benefits to estimate whether the aggregated prices of a particular open source distributor are too high.[252] Admittedly, this is an

[250] *See* Master End-User License Agreement for Microsoft Software, *supra* note 31 (giving the anti-open source license provisions of a Microsoft license).

Even a large software company generally supportive of open source, Oracle, determined that it needed anti-open source provisions to guard against infectious terms for some of its software components. Oracle Technology Network, Oracle Technology Network Developer License Terms, *at* http://otn.oracle.com/software/htdocs/devlic.html (last visited Dec. 27, 2004). The specific provision is as follows.

> [Y]ou may not develop a software program using an Oracle program and an Open Source program where such use results in a program file(s) that contains code from both the Oracle program and the Open Source program (including without limitation libraries) if the Open Source program is licensed under a license that requires any "modifications" be made freely available. You also may not combine the Oracle program with programs licensed under the GNU General Public License ("GPL") in any manner that could cause, or could be interpreted or asserted to cause, the Oracle program or any modifications thereto to become subject to the terms of the GPL.

Id.

[251] *See supra* notes 192-99 and accompanying text.

[252] From the perspective of a user deciding whether to use open source code, a number of factors are at issue. FINK, *supra* note 6, at 95-105 (listing and discussing over a dozen cost factors).

One factor not listed by that commentator is opportunity cost. Taking someone else's open source software and deploying it in one's organization may mean forgoing any future imprecise process. But, within broad divisions, it should allow user buying power to discipline open source distributors.

Many other factors could influence the interrelationship among infectious terms, source code availability, and the anti-royalty provision of open source licenses. These include the type of software; whether it is a platform technology; its degree of susceptibility to increased value due to network effects; competition in the application space from both proprietary products and other open source software; a host of technological factors; and factors related to the open source development community's norms and practices.[253] Within this haze of additional factors, there is some possibility that infectious terms benefit the other key open source terms. However, the reasons discussed above show that there are limits to these beneficial effects.

The same limits hold true for the first argument for infectious terms: that they reinforce open source software development, adoption, and conversion of other software. The last claim seems particularly unlikely, given the size and scale of the proprietary software industry.

There are other reasons to think that the posited benefits of infectious terms are limited. Many other aspects of the innovative and clever GPL have been imitated. But infectious terms have been the GPL's least imitated feature. A key certification program for open source licenses does not require infectious terms.[254] Some open source projects, such as Apache, are successfully without not only infectious terms, but also without most other key open source license conditions. This indifferent attitude to infectious terms introduces the next section, the potential problems arising from broadly stated infectious terms of expansive scope.

opportunity to commercialize improvements based on the original open source code. This opportunity cost is greater the stronger the infectious terms. Infectious terms make it more difficult to partition the improvements in ways that might escape infection. In addition, such partitioning may drive the programmers to suboptimal software design approaches in order to avoid infection. Thus, an organization might weigh these potential costs of partitioning and designing around infection in its decision. It is certainly conceivable that in some cases, these costs and the opportunity cost will outweigh the no-royalty and other perceived benefits of open source.

[253] *See* Vetter, *supra* note 7, at 627, 630-31 (describing the norms and practices of open source software development, and describing the influence of programmer reputation on the development process).

[254] *See* OSD, *supra* note 48, §§ 1-3, 7-9 (none of the relevant sections of the Open Source Definition discuss infectious terms, although the OSD does acknowledge that copyright's derivative work right will apply for modified, extended, or intermixed works).

2. Problems with Infectious Terms

Uncertainty in the locus of infection is the hallmark difficulty for broad infectious terms. This manifests in several ways. The risk-averse are less likely to adopt open source software with such terms. The uncertainty impedes useful combinations between open source code and proprietary code. And legal risks such as indirect liability or even copyright misuse may arise or be exacerbated by expansive infectious terms. Moreover, the uncertainty has costs of its own arising from increased transaction costs and difficulty in determining the scope of rights in the code.

As the efficacy model discussion surrounding Figure 6[255] illustrates, both the technological and legal continuum underlying infectious terms do not readily admit to precise boundaries. The GPL's infectious terms compound the difficulty. Its rubric of the "whole" provides little guidance to eliminate the uncertainty. Its two safe harbors cover some situations, but leave much unspecified. Any broadly specified infectious terms that incorporate copyright's derivative work standard as a measure of the terms' scope will similarly suffer.

These uncertainties create risks daunting to some who might otherwise adopt open source software. A growth opportunity for GNU/Linux is corporate information technology departments.[256] These groups represent an important purchasing force in computing, but infectious terms elevate their risk of using open source software. Such terms create a need to internally partition and segregate code to assure that the open source software does not infect existing code internally developed or provided by third parties.[257]

[255] *See supra* Part V.B.

[256] Both IBM and Red Hat are marketing GNU/Linux in the corporate information technology market. *See* William M. Bulkeley, *Out of the Shadows: Open-source Software is not Only Becoming Acceptable; It's Also Becoming a Big Business*, WALL ST. J., Mar. 31, 2003, at R6 ("[M]ajor companies like IBM, Sun and Hewlett-Packard Co., awakened to the profit opportunities in providing hardware and services linked to free software, are paying some of their programmers to work on Linux and other open-source software."), *available at* 2003 WL-WSJ 3963322; *supra* note 208 (describing Red Hat's efforts to market GNU/Linux for enterprise computing).

[257] Whether internal partitioning and similar precautions are *per se* required is an interesting question. Under the GPL, redistribution of the software triggers the key open source obligations. Since the GPL specifically uses the word "distribution," copyright's public distribution right is in play. There is some ambiguity to the degree of "public-ness" necessary to violate copyright's distribution right. NIMMER, *supra* note 28, § 8.11[A]. Suppose an internal user blithely intermixes open and closed code, assuming that there would never be a distribution triggering the open source conditions. However, corporate managers

Moreover, proprietary vendors are reacting to infectious terms by erecting licensing safeguards to prohibit intermixing with open source.[258] Thus, information technology purchasers must respond to these requirements. Although infectious terms are not the crux of the *SCO* case,[259] there is anecdotal evidence that the infringement risk publicized by SCO is slowing the rate at which corporate information technology departments adopt and deploy GNU/Linux.[260] Risk perceptions from infectious terms have similar effects. Also, a key GNU/Linux competitor, Microsoft, touts infectious terms as a key risk factor for corporate adoption of open source software.[261] Given all these influences, broad infectious terms impede open source deployment and to some degree slow its adoption rate.

The FSF has argued against its own Lesser GPL license because that license puts "popularity" above reinforcing advantages among open source programmers.[262] If one takes "popularity" to mean deployment in mixed computing environments using both open and closed code, the argument presupposes that the benefits from network effects for interoperating with proprietary code are less valuable than the gains to be had by cloistering (via infectious terms) open source code to only open source environments. This argument implicitly acknowledges that infectious terms will result in less adoption of open source. It justifies this result with benefits arising from

want to minimize risk and allow for future contingencies. Thus, such blithe intermixing could put an information technology department in a difficult position if later a distribution triggering copyright's distribution right became necessary. Further, corporate technology managers now face anti-intermixing license terms from some proprietary software vendors. *See supra* notes 31, 250 (describing licenses from proprietary software vendors with anti-open-source provisions). These license provisions require a buffer between open and closed software, limiting beneficial interoperability.

[258] *See supra* notes 31, 250.

[259] *See supra* notes 17-18, 20-22 and accompanying text.

[260] *Head to Head*, FIN. SECTOR TECH., Nov. 30, 2003 (reporting that a key information technology analyst group was advising corporate customers "to delay deployment of Linux in important systems until the case was settled"), *available at* 2003 WL 61100882; Joseph Menn, *SCO Suit May Blunt the Potential of Linux*, L.A. TIMES, June 6, 2003, *available at* 2003 WL 2417592.

[261] Microsoft Corp., Some Questions Every Business Should Ask About the GNU General Public License (GPL), *at* http://www.microsoft.com/korea/business/downloads/licensing/Gpl_faq.doc (last visited Aug. 7, 2003, but later found to be unavailable at this address) (on file with author) (discussing in question nine the risks perceived by Microsoft arising from the GPL's "viral" nature).

[262] Why-Not-LGPL, *supra* note 153.

creating a critical mass of high-quality open source software. How the balance factors out does not admit to an empirical answer, although one wonders whether GNU/Linux would have achieved its critical mass without an explicit safe harbor for user programs merely running on the operating system making normal system calls.

The second difficulty arising from uncertain infectious terms is that they impede useful combinations between open source code and proprietary code, thereby inhibiting interoperability and compatibility. The efficacy model of Figure 6[263] and the examples styled from the *Progress Software*[264] case both illustrate this. Situations like the *Progress Software* case suggest direct impedance of beneficial interoperability.[265] To some degree, the database engine was designed to operate with multiple storage modules. On one hand, a compatibility capability was provided. But, on the other, infectious terms eliminated its usefulness. The efficacy model shows that infectious terms measured by the derivative work right will locate in a fuzzy outer zone of intermixed and coupled "other" software. When such "other" software is proprietary, the result may be that the user foregoes intermixing, possibly foregoing beneficial interaction between the code. Or, the infectious terms might prod a user to couple the software in less-than-optimal ways in order to give a wide berth to the infectious zone.

Another example of this effect is interoperability within the Linux kernel between core GPL protected kernel code and loadable modules,[266] some of which are proprietary. The Linux kernel is modular. Recall that the kernel is the first and most powerful program that executes when the computer is powered on. It controls the computing resources and parcels these resources to other processes that need them to do work. The kernel has a capability to load modules, i.e., other containers or units of executable code that, in essence, become part of the special and powerful process called the kernel.

Within the kernel, the GPL's regular provisions apply. Thus, its infectious terms threaten to capture any loadable kernel modules that are not licensed under the GPL. This has been a source of friction and uncertainty during the evolution of the kernel and continues to engender questions today. Even with the GPL's infectious terms, some kernel modules are not covered by the GPL, yet are regularly used in GNU/Linux. Furthermore, there have

[263] *See supra* Part V.B.

[264] 195 F. Supp. 2d 328 (D. Mass. 2002).

[265] *See supra* notes 191-93, 195 and accompanying text.

[266] The issues with loadable kernel modules, or LKMs, are discussed *supra* note 161.

been no infringement actions. One example given by Linus Torvalds is a driver for a particular filesystem.[267] It is not GPL protected, but Torvalds does not consider it a derived work because it existed in full before the Linux kernel was developed, and was only later adapted to work with the kernel.[268] This kernel module is a good example of "other" software that might be thought to fit into the GPL's identifiably independent and separate works safe harbor.

Interoperability and compatibility within the Linux kernel is one of the most important places to have such features. While the GPL's infectious terms have thrown a cloud on these features, other factors have compensated. In some cases, the FSF's predictions have come true – kernel module providers simply succumb to the GPL's terms.[269] In other cases, some kernel modules retain non-GPL terms, but have encountered minimal resistance.[270] The fact that these situations have not resulted in infringement suits from kernel contributors is most likely attributable to the deft management touch of Linus Torvalds and his "lieutenants," who direct the Linux kernel development effort and facilitate beneficial community relations.

The third difficulty arising from uncertain infectious terms is that they may engender or exacerbate legal risks such as indirect liability or even, as suggested by at least one commentator, copyright misuse.[271] As to the latter, if the GPL's "whole" is enforced as a contract, perhaps because the software presented the GPL and implemented an "I Accept" assent from the user, the question becomes: what is meant by the "whole"? In particular, the specter of misuse becomes at least remotely plausible when one considers situations of apparent imbalance: a ten page open source program "infecting" a one hundred (one thousand?) page proprietary program. The idea that any other

[267] Torvalds Email, supra note 162. Torvalds elaborates as follows:

> There are (mainly historical) examples of UNIX device drivers and some UNIX filesystems that were pre-existing pieces of work, and which had fairly well-defined and clear interfaces and that I personally could not really consider any kind of "derived work" at all, and that were thus acceptable. The clearest example of this is probably the AFS (the Andrew Filesystem).

Id.

[268] *Id.*

[269] *See supra* Part V.A.

[270] *See supra* Part V.A.

[271] *See* McClelland, *supra* note 229.

aspects of a typical open source software license create misuse seems farfetched.[272] However, the potential sweep of infectious terms might give one pause before dismissing that notion. As to indirect copyright liability, it requires a direct act of infringement. Broad infectious terms increase the likelihood of a direct infringing act, thus increasing the infringement risk of those distributing open source and proprietary components that can be intermixed or coupled. This risk still may be small, but at the margin, broad infectious terms, as compared to narrow terms, elevate the risk.

Given these difficulties that arise from broad infectious terms of expansive reach, the next section argues that on balance their incentive effects are not beneficial. Although there may be some limited justifications for infectious terms, they do not offer a compelling case given the alternative of specifying infectious terms more tightly.

3. Striking a Balance: Limited Quarantine for Infectious Terms?

In balancing any purported benefits of infectious terms against the problems of broad, uncertain terms, one factor is that such terms are unlikely to convert any appreciable percentage of proprietary software to open source terms. This is not to say that the open source approach will not continue to grow and impact both open source software and proprietary software. Some proprietary software vendors have employed open source approaches to

[272] The other key open source terms should not raise copyright misuse concerns when applied voluntarily by a programmer to her code. It is difficult to conceive how "donation" of a programmer's code extends the programmer's control in a way that implicates the antitrust-like concerns animating copyright misuse. The programmer is asking that the source code be available and that its use and redistribution be without royalties. This condition applies to her code (and only her code - if there are no infectious terms). This is not so dissimilar from making a donation to one's law school alma mater with the condition that half the donation be used for student scholarships, and that the other half is used to renovate a building that will henceforth bear the name of the donor. There is an impact on competition resulting from the donation - other law schools have less attractive buildings. But, this is an impact on competition that our history and policy has not found troubling, with the potential exception of the law's jealousy that tax exempt status for the donation and the recipient not be abused. By my discussion, I do not mean to say that no other terms in an open source license could ever raise misuse concerns. Rather, the key provisions that keep the software "open source" should not be thought to do so, especially when applied only to the contributed code.

At bottom, and aside from infectious terms, copyright misuse is a proposition that fits oddly with the gift-based peer production model behind open source software. *See* McGowan, *Legal Implications*, *supra* note 4, at 288 (discussing, in the context of fair use, that the fair use factor, "effect on the market," has less meaning for open source, and that courts should instead inquire as to the effect of unauthorized infringement on the production model generating the open source software).

varying degrees.[273] Furthermore, open source software's flagship applications, such as GNU/Linux, continue to gain market share. Thus, the other benefits of the collaborative open source development model paint a growing and positive role for open source software.

Too many factors stand in the way of infectious terms converting the large installed base of existing proprietary code. By any aggregate measure, proprietary code still dwarfs open source code. Many vested interests permeate proprietary code. Most of it is controlled by corporate entities with profit maximization mandates. These mandates, and the momentum of the status quo, make it difficult for these entities to conceive of business models based on open source. Even if an entity wanted to adopt an open source business model, it would want to do so on its own terms, and not by a surprise infection.[274]

[273] Some proprietary vendors engage in "dual-licensing." Many in the open source community dislike this practice, but a number of proprietary vendors have built successful business models from dual licensing. One example is the Qt product from a company called Trolltech. Trolltech, Licensing Overview, *at* http://www.trolltech.com/products/licensing.html (last visited Dec. 27, 2004) (showing that the Qt product is available under a commercial or under a free use license). Qt is a product for programmers. It contains, among other items, libraries of routines that programmers can use in their code. The dual license approach keeps two parallel versions of the product, one for free use and one for commercial use.

A rough characterization of the dual license for Qt is as follows. Under the Qt Public License, a programmer can take and use the free version, but cannot distribute a commercial product with it. *See* Trolltech, QPL License (annotation), *at* http://www.trolltech.com/licenses/qpl-annotated.html (last visited Dec. 27, 2004). Moreover, for any modifications the programmer makes to the free Qt product, the programmer grants a license to Trolltech to use the programmer's modifications in a commercial version so long as the modifications also remain in the free version. *Id.* Thus, the commercial version can siphon functionality from the Qt free version. Further, anyone who wants to use the free version as "free and open source software" can do so. *See id.* The Qt free software license is in essence a modified GPL with additional dual-licensing conditions.

[274] A footnote to one of the most celebrated events in the growth of the open source movement offers a perspective on how entities might make the switch to open source. In 1998, Netscape stunned the software world by announcing that it would release an open source version of its web browser. Jim Hamerly et al., *Freeing the Source: The Story of Mozilla*, *in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION, *supra* note 2, at 197-206. At that time, the Netscape browser was the leader in its field. Netscape created two licenses, one for the existing code base, the Netscape Public License (NPL), and one for new files, the Mozilla Public License (MPL). Mozilla Public License, *supra* note 48; Netscape Public License Version 1.1 [hereinafter NPL], *available at* http://www.mozilla.org/MPL/NPL-1.1.html (last visited Dec. 27, 2004). A "frequently asked

Some entities are proactively modifying their license agreements for redeployable software components in order to immunize them from infection. This counter-reaction is not unexpected given their corporate duties to protect and shepherd assets. Another counter-reaction appears in the intellectual property due diligence accompanying the sale of entities or their assets. During the last few years, anti-open-source provisions have appeared in many of the form agreements for such transactions. These considerations suggest that the benefits of open source development[275] are attractive to proprietary software vendors, but that these vendors refuse to accept coercive infectious terms.

Rather than adopt broad, uncertain scope license terms that consider intermixed and coupled software, licenses should draw as bright a line as possible to cabin the scope of infection.[276] Saying nothing might be as unclear as a broad phrase like the GPL's "whole." Saying nothing could leave the scope of the open source permission to hinge on copyright's derivative work right for modified, intermixed, or coupled software. A bright line standard could be expressed in several ways: (i) in terms of specific technical scenarios where there is to be no infection; (ii) in terms of

questions" document discusses both licenses, describing Netscape's reasons for creating two licenses: the existing code was encumbered with intellectual property and contractual obligations, so the NPL gave Netscape rights to use the open source contributions, but Netscape had no such rights in separate and new code contributed under the Mozilla License, or MPL. Mozilla, Netscape Public License FAQ, *at* http://www.mozilla.org/MPL/FAQ.html (last visited Dec. 27, 2004) [hereinafter Netscape FAQ]. Netscape described the matter as follows:

> We share some code between clients and servers, and we wanted to make sure that we could make changes to that code and take advantage of those changes in our server products without having to release those products under the NPL as well. . . . And finally, we have a number of outstanding contracts to supply source code with which we need to remain in compliance.

Id.

[275] Some proprietary vendors gravitate to open source development in order to use the dual-license mechanism to benefit commercially from community programming efforts, yet also contribute something back to the community by making a free version of the software available. *See* FINK, *supra* note 6, at 40-41, 180-82 (describing dual-licensing); *supra* note 273. There are a number of potential business models applicable to open source. FINK, *supra* note 6, at 175-76. These include using open source software to enable hardware sales, the core of IBM's strategy. The list also includes dual-licensing, service and support such as that provided by Red Hat, and others. *Id.* Finally, many companies are studying or deploying the far-flung, collaborative development style that has evolved around, and as a result of, the open source software licenses. *Id.* at 137-42.

[276] *See supra* note 227 and accompanying text.

specific goals, such as immunizing from infection for defined types of interoperability or compatibility; or (iii) by reference to some form of digital rights expression.[277] Any of these methods, or other methods, that quarantine the scope of infection would benefit the open source software distributed under such a license.

Linus Torvalds' decision to clearly state that user programs calling the Linux kernel were not infected is an example of the first method.[278] Another example is the guidance the Mozilla open source project issued with its two licenses when Netscape converted its web browser into an open source project. There were two licenses because Netscape had obligations that would not permit it to put the existing code into a typical open source license.[279] Thus, it provided the source and used a first license to apply the open source conditions to everyone else, but reserved rights to Netscape to use contributed modifications in its proprietary products.[280] While such was against the spirit of open source development, to its credit, Netscape established a second license without these special rights for new code contributions.[281]

For my analysis, the key point is the precision with which the licenses defined the scope of the permission and its application. The Netscape licenses and the "frequently asked questions" document specify what is considered a modification to the existing code.[282] There are no infectious

[277] Classically, the term has been digital rights management, rather than digital rights expression. *See* Symposium, *Edited & Excerpted Transcript of the Symposium on the Law & Technology of Digital Rights Management*, 18 BERKELEY TECH. L.J. 697, 732-34 (2003) [hereinafter DRM Transcript] (giving comments of Professor Lessig, arguing that digital rights expression is different from digital rights management, with the former perhaps being a subset of the latter, but with the latter providing the means to control the copyright holder's expressed allowed uses for the digital work). Regardless of the terminology, these are measures that use technological self help to enforce the rights of copyright holders and conditions they have keyed to those rights. Burk, *supra* note 229, at 1100-02; *see also* Dan L. Burk & Julie E. Cohen, *Fair Use Infrastructure for Rights Management Systems*, 15 HARV. J.L. & TECH. 41, 42-43, 47-54 (2001) (examining whether "rights management systems [can] be designed and implemented in a way that preserves the traditional copyright balance," and reviewing the issues surrounding rights management information).

[278] *See supra* note 156 and accompanying text.

[279] Netscape FAQ, *supra* note 274.

[280] *Id.*

[281] *Id.*

[282] Mozilla Public License, *supra* note 48, § 1.9; NPL, *supra* note 274 (same). The provision is set forth below.

terms in either license in order to maximize the opportunity for compatibility between the new contributions and the existing code base. New files are under the second license "even if the new file is called or referenced by changes [a programmer] made in [an old file]."[283]

Specified interoperability goals are another way to cabin infectious terms. This was implicit in the Netscape/Mozilla licenses, but explicit in the parole "frequently asked questions" document. These goals could be expressed in an open source license with infectious terms. The terms would then have a safe harbor that is more useful than the GPL's safe harbors. In addition, this approach correlates to copyright's fair use doctrine as applied to reverse engineering.[284]

The third suggested method to limit infectious terms is to use the technology itself. Recently, commentators have re-expressed the traditional paradigm of digital rights management as "digital rights expression."[285] Digital rights expression is best exemplified by the Creative Commons project,[286] which is itself inspired from the open source software movement. Creative Commons provides licenses for content creators to apply to their work when they wish to share a work. While the licenses do not apply to software, they can be encoded in the digital content so the content can

> "Modifications" means any addition to or deletion from the substance or structure of either the Original Code or any previous Modifications. When Covered Code is released as a series of files, a Modification is:
>
> A. Any addition to or deletion from the contents of a file containing Original Code or previous Modifications.
>
> B. Any new file that contains any part of the Original Code or previous Modifications.

Mozilla Public License, *supra* note 48, § 1.9.

[283] Netscape FAQ, *supra* note 274.

[284] *See supra* notes 235-36 and accompanying text.

[285] *See* DRM Transcript, *supra* note 277, at 732-33 (comments of Professor Lessig).

[286] Creative Commons, Frequently Asked Questions, *at* http://creativecommons.org/faq (last visited Dec. 27, 2004). The Creative Commons licenses "do not make mention of source or object code;" thus the Creative Commons organization recommends that creators seeking to publish open source software use an open source software license. *Id.* Creative Commons offers "the public a set of copyright licenses free of charge. These licenses will help people tell the world that their copyrighted works are free for sharing -- but only on certain conditions." *Id.* Content producers can select from a menu of possible licenses, each specifying a different set of conditions for sharing the content. Thus, Creative Commons provides tools with which individual work-holders can apply an open source approach of their choosing to make their work available for sharing. It seeks to generally enable and apply the open source approach beyond software.

indicate the degree of sharing allowed.[287] This innovation could be retrofit into open source software to express the boundary of infectious terms.

Something similar is starting to occur within the Linux kernel, although not in a way that limits the GPL's infectious terms. When executable object code intermixes in the runtime environment, both sets of code must exchange information about the subject of their interactions. Each makes the other aware of certain aspects of itself. This exchange between the Linux kernel (and any previously loaded kernel modules) and a loadable kernel module occurs in a specific, technically defined way using capabilities provided by the kernel. These capabilities include an exchange that discloses the license used by the module.[288] Recent enhancements to this mechanism allow earlier loaded modules to withhold their information from later loaded modules that do not use a GPL compatible license.[289] This in essence is an anti-interoperability mechanism. Some developers will not want their kernel modules to interact with non-GPL compatible licenses.

This subsection has reviewed three mechanisms to make infectious terms more "bright line": greater technical specificity in the license; expression of safe harbors for interoperability and compatibility; and digital rights expression. All three of these, or any combination of them, would help quarantine broad infectious terms of expansive scope. Such terms, on balance, despite some potential benefits, are unlikely to benefit open source software due to the incentive effects they engender. They are unlikely to convert proprietary software in appreciable mass to open source software, they inhibit beneficial interaction between the two types of code, and they

[287] *Id.*

[288] Tux.org, The linux-kernel mailing list FAQ, "What does it mean for a module to be tainted?" § 1(18), *at* http://www.tux.org/lkml (last visited Dec. 27, 2004) (describing that the Linux kernel keeps a list of GPL-compatible licenses, and that when new kernel modules are loaded they have an identifier indicating their license which the load process uses to compare the identifier to the list, and if the module's license does not appear on the list the loader delivers certain warnings).

[289] *Id.* at "What is this about GPLONLY symbols?" § 1(19). The explanation below describes the mechanism that allows Linux kernel module programmers to withhold interoperability information from GPL incompatible modules.

> [This capability is to] allow choice for developers who wish, for their own reasons, to contribute code which cannot be used by proprietary modules. Just as a developer has the right to distribute code under a proprietary license, so too may a developer distribute code under an anti-proprietary license (i.e. strict GPL).

Id.

create additional risks for open source users. The uncertainty of such terms is too high, raising costs, inhibiting interoperability, and thus impeaching their efficacy. Moreover, one can posit examples, styled from the *Progress Software*[290] case, where open and closed code could co-exist. After isolating the infectious terms, the other open source license conditions could be satisfied for the open source portion of the mixed-code whole. They could co-exist as an interoperating, cooperative whole.

VI. CONCLUSION

The questionable efficacy of infectious terms arises from the need to cabin their scope along two interrelated continuums: copyright's reproduction and derivative work rights, and the technical progression of modifying, extending, intermixing and coupling open source and proprietary software. The GPL's infectious "whole" has been my foil and prime example. It uses copyright to create the infectious mechanism, specifically, the derivative work right. But the tensions arising from infectious terms apply whenever they have expansive breadth and reach that inhibit beneficial software interoperability and increase risks for open source users.

This results in misaligned incentives for optimal coexistence between open source and proprietary software. While infectious terms may have a role to play in open source licensing, broad and far-reaching terms, on balance, impair open source software. The GNU/Linux flagship open source application offers the foremost example of this. A decision early in the Linux kernel project to specify a key safe harbor from otherwise broad infectious terms allowed certainty to prevail. From this came confidence important to the market growth of the operating system. Similar actions in open source licensing generally would help ensure that the licenses themselves spread the optimal incentives without creating unnecessary resistance to the open source movement.

[290] Progress Software Corp. v. MySQL AB, 195 F. Supp. 2d 328 (D. Mass. 2002).