The Collaborative Integrity of Open-Source Software

Greg R. Vetter[*]

This Article analyzes legal protection for open-source software by comparing it to the venerable civil law tradition of moral rights. The comparison focuses on the moral right of integrity, with which one may object to mutilations of her work, even after having parted with the copyright and the object that embodies the work. The parallel apparatus in open-source licensing is conditional permission to use a copyrighted work. The conditions include that source code be available and that software use be royalty-free. These conditions facilitate open-source collaborative software development. At the heart of both systems is the right for creators to control the view that a work presents. In the open-source system, this is the Collaborative Integrity of open-source software. The history and legacy of moral rights help us better understand Collaborative Integrity in open-source software. The right of integrity in some international jurisdictions may apply to software, thus raising questions whether it hurts or helps open-source software. Building from these insights, this Article evaluates whether the Collaborative Integrity in open-source software deserves protection as a separate right, just as the right of integrity developed separately from pecuniary copyright in civil law jurisdictions.

---

[*]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: http://www.law.uh.edu/faculty/gvetter. It is relevant to this Article 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 and Kristin Brown for comments, suggestions, discussion, and research assistance on this Article. Also, my thanks to workshop participants at UHLC, Marquette University Law School, and the Louisiana State University Law Center for helpful discussion of some of the issues in this Article. In addition, I am thankful for exceptionally capable support provided by the staff of the UHLC's John M. O'Quinn Law Library, including Peter Egler and Nicole Evans, and by the UHLC's Legal Information Technology Department, including Robert A. Brothers and Chad J. Kitko. I am also indebted to the Barco Law Library at the University of Pittsburgh School of Law, and to Marc Silverman for graciously making the Barco Law Library available for my research. My special thanks to Craig Joyce, Mike Madison, Mark Lemley, and David McGowan for helpful comments. I wish to dedicate this Article, my first as a law professor, as with all my efforts, to my spouse and life partner, Pamela "Christy" Parham-Vetter, M.D.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

I. INTRODUCTION

> *When I finished grad school in computer science I went to art school to study painting. A lot of people seemed surprised that someone interested in computers would also be interested in painting. They seemed to think that hacking and painting were very different kinds of work—that hacking was cold, precise, and methodical, and that painting was the frenzied expression of some primal urge.*
>
> *Both of these images are wrong. Hacking and painting have a lot in common. In fact, of all the different types of people I've known, hackers and painters are among the most alike.*
>
> *What hackers and painters have in common is that they're both makers. Along with composers, architects, and writers, what hackers and painters are trying to do is make good things.*[1]

Since the dawn of computing a half-century ago, software has hidden its human-readable source code in non-readable "object code" that only the computer can interpret. In part, software development technology made object code the preferred mode of distributing and running software. The law, however, reinforced this preference. Before the advent of software copyright protection, developers used trade secret law to protect software, relying on non-readable object code to protect the secret.[2] Even after the advent of

---

[1]Paul Graham, *Hackers and Painters*, *at* http://www.paulgraham.com/hp.html (May 2003) (displaying Web site self-published essay from guest lecture given by Mr. Graham at Harvard). Among programmers, "hacker" means a skilled computer enthusiast or programmer, and is generally a favorable label in that community, but the popular press often uses the term derogatorily to designate one who seeks to "gain unauthorized access to computer systems for the purpose of stealing and corrupting data." Webpodedia, *Hacker*, *at* http://www.webopedia .com/TERM/h/hacker.html (last visited Jan. 26, 2004).

[2]Peter S. Menell, *Envisioning Copyright Law's Digital Future*, 46 N.Y.L. SCH. L. REV. 63, 74 (2002–03); Thomas M. Pitegoff, *Open Source, Open World: New Possibilities for Computer Software in Business*, BUS. L. TODAY, Sept.–Oct. 2001, at 52 ("While source code is protected by copyright and may be protected by patent, in practice source code is protected by trade secrecy. Even if the source code copyright is registered . . . , only a small portion of the code need be filed on registration, while most of it typically remains secret."); Pamela Samuelson, *CONTU Revisited: The Case Against Copyright Protection for Computer Programs in Machine-Readable Form*, 1984 DUKE L.J. 672, 673 ("[S]oftware manufacturers generally market only machine-readable forms of programs, thereby withholding not only their ideas, but much or all of the manner in which those ideas are expressed."). *See also* PETER WAYNER, FREE FOR ALL: HOW LINUX AND THE FREE SOFTWARE MOVEMENT UNDERCUT THE HIGH-TECH TITANS 9 (2000) (noting commercial software vendor's preference to guard source code for trade secret protection), *available at* http://www.wayner.org/books/ffa (last visited Feb. 4, 2004); Chris DiBona et al., *Introduction* to OPENSOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION 1, 2 (Chris DiBona et al. eds., 1999) [hereinafter OPENSOURCES] (noting commercial software vendor's preference to guard source code for trade secret protection).

software copyright protection, lawyers advised clients to conceal source code to prevent others from copying and infringing it.

Software development is changing.[3] The Internet allows far-flung development teams to collaboratively create software.[4] Market forces demanding interoperability and standardization encourage disclosure of traditionally concealed software elements, such as data structures, code, files, and other internal elements. Traditional intellectual property law—copyright, trade secret, and patent—has failed to keep pace. But a recent innovative use of copyright and licensing law has established a new alternative regime: open-source software.[5] Open-source programmers share source code royalty-free and collaborate in ad hoc, self-organizing, intercorporate units to develop

---

[3]*See* FREDERICK P. BROOKS, JR., THE MYTHICAL MAN-MONTH: ESSAYS ON SOFTWARE ENGINEERING, ANNIVERSARY EDITION 278–89 (1995) (recounting changes in software engineering and software industry during prior twenty years and projecting future changes).

[4]David McGowan, *Legal Implications of Open-Source Software*, 2001 U. ILL. L. REV. 241, 251 (2001); Audris Mockus et al., *Two Case Studies of Open Source Software Development: Apache and Mozilla*, 11 ACM TRANSACTIONS ON SOFTWARE ENG'G AND METHODOLOGY, No. 3, at 309, 310 (2002). *See generally* Yochai Benkler, *Coase's Penguin, or, Linux and the Nature of the Firm*, 112 YALE L.J. 369, 383, 434–36 (2002) (postulating formal model describing collaborative peer production for information products, and conditions under which the model will tend toward such peer production, including the necessity of organizing communications among peers, typically through the Internet).

[5]*See* James Bessen, *What Good Is Free Software?*, *in* GOVERNMENT POLICY TOWARD OPEN SOURCE SOFTWARE 12, 13 (Robert W. Hahn ed., 2002), *available at* http://aei-brookings.org/admin/pdffiles/phpJ6.pdf (last visited Feb. 4, 2004); Patrick K. Bobko, *Open Source Software and the Demise of Copyright*, 27 RUTGERS COMPUTER & TECH. L.J. 51, 75–76, 80–81 (2001); Robert W. Gomulkiewicz, *How Copyleft Uses License Rights to Succeed in the Open Source Software Revolution and the Implications for Article 2B*, 36 HOUS. L. REV. 179, 181–82, 185–86 (1999); Marcus Maher, *Open Source Software: The Success of an Alternative Intellectual Property Incentive Program*, 10 FORDHAM INTELL. PROP. MEDIA & ENT. L.J. 619, 620, 637–38 (2000); McGowan, *supra* note 4, at 242–43; Stephen M. McJohn, *The Paradoxes of Free Software*, 9 GEO. MASON L. REV. 25, 28 (2000); Shawn W. Potter, *Opening up to Open Source*, 6 RICH. J.L. & TECH. 24, ¶¶ 7–8 (Spring 2000), *at* http://law.richmond.edu/jolt/v6i5/article2.html#h1; Daniel B. Ravicher, *Facilitating Collaborative Software Development: The Enforceability of Mass-Market Public Software Licenses*, 5 VA. J.L. & TECH. 11, at *30, *63 (2000), *available at* http://www.vjolt.net (last visited Feb. 4, 2004); Ira V. Heffan, Note, *Copyleft: Licensing Collaborative Works in the Digital Age*, 49 STAN. L. REV. 1487, 1491–92 (1997).

software.[6] They do so under legal norms specified in a generally applicable license.[7] Within these legal norms, there is a set of license conditions I call the open-source approach.[8] Beyond the legal norms, however, community norms and a unique open-source culture influence the collaborative effort.[9] Thus, from a broad view, open-source software is research methodology, production method, business model, social statement, and industrial rebellion, all rolled into one "movement."

---

[6]*See* Open Source Initiative, *OSI Position Paper on the SCO-vs.-IBM Complain*, *at* http://www.opensource.org/sco-vs.ibm.html#seismic (last visited Jan. 26, 2004) (describing use of small distributed groups for "Internet style" of coordinating large programming teams). Raymond contrasts the "Internet style" with traditional software development.

> Since the 1960s, the Internet and Unix hackers had been pioneering a style of software engineering which reversed the premises of industrial software development. Instead of centralization in large programming teams, the Internet style used small distributed programming groups. Instead of process control and hierarchy, the Internet style used peer review and open standards. Most importantly, the Internet style abolished secrecy in favor of transparency and what came to be called "open-source" code.

Id. (citation omitted).

[7]*See* Joseph Scott Miller, *Allchin's Folly: Exploding Some Myths About Open Source Software*, 20 CARDOZO ARTS & ENT. L.J. 491, 496–97 (2002); Pitegoff, *supra* note 2, at 52–54; Jason B. Wacha, *Open Source, Free Software, and the General Public License*, COMPUTER & INTERNET LAW, Mar. 2003, at 20, 20–22 (describing open-source approach as "one of the fastest growing, ever adapting, and most commonly misunderstood licensing schemes in the world").

[8]As I use the term "open-source approach" in this Article, I mean to refer only to software and its source code. While I briefly note later that the open-source approach may apply beyond software to content generally, *see infra* Part III.B.2, this interesting extension of the approach is not my focus. There are a variety of licenses that define the legal norms, and significant variance among widely used licenses. *See* GNU Project—Free Software Foundation, *Various Licenses and Comments about Them*, *at* http://www.gnu.org/licenses/license-list.html (last visited Feb. 4, 2004) (listing and discussing licenses, including General Public License ("GPL"), one of the most popular licenses authored by founder of Free Software Foundation); Open Source Initiative, *The Approved Licenses*, *at* http://opensource.org/licenses/index.html (last visited Jan. 26, 2004) (listing and discussing licenses); Bruce Perens, *The Open Source Definition*, *in* OPENSOURCES, *supra* note 2, at 180–85 (describing, comparing and contrasting several popular licenses, noting that some substantial ways in which they differ are whether (1) they require future distributions of modified or unmodified software to include source code, (2) they prohibit users and redistributors from charging royalties for use, and (3) they require continued propagation of the same license terms for software or modified works based on software); Mark H. Webbink, *Open Source Software—Bridging the Chasm*, *in* 22ND ANNUAL INSTITUTE ON COMPUTER LAW 663, 674 (2002) (listing various licenses) (Mr. Webbink is senior vice president and general counsel for Redhat, Inc., a leading Linux distributor). As I will describe, what I call the open-source approach requires, among these differences, that the source code go with the software, that royalties are prohibited, and that the same terms propagate; which is most closely aligned with the GPL as compared to other licenses.

[9]Robert W. Hahn, *Government Policy Toward Open Source Software: An Overview*, *in* GOVERNMENT POLICY TOWARD OPEN SOURCE SOFTWARE 1, 2 (Robert W. Hahn ed., 2002), *available at* http://aei-brookings.org/admin/pdffiles/phpJ6.pdf (last visited Jan. 27, 2004); Maher, *supra* note 5, at 631–35; McGowan, *supra* note 4, at 260–61.

The social benefit of the movement is not in question. Much of the Internet runs on open-source software.[10] Many companies have open-source business strategies.[11] An open-source operating system, Linux,[12] competes with Microsoft in certain markets.[13] Open source has energized the debate about software quality while reliability problems still challenge traditional software.[14] Accessibility to the source code plays a key role in providing these social benefits. This accessibility is created and enforced by the innovative open-source approach.[15]

The open-source movement, therefore, to some degree rests on a nascent legal foundation, or at least a nascent use of preexisting legal mechanisms. In this Article, I comparatively assess this foundation. The assessment contrasts

---

[10]Joseph Feller et al., *Making Sense of the Bazaar: 1st Workshop on Open Source Software Engineering*, 26 ACM SIG SOFTWARE ENG'G NOTES No. 6, at 51, 51 (Nov. 2001) ("Many Open Source products (the Apache HTTP server, . . .) are category leaders in the Internet application space, and others (Linux, . . .) are becoming increasingly popular as components in enterprise computing architectures."); Wacha, *supra* note 7, at 20.

[11]Pitegoff, *supra* note 2, at 54.

[12]Since this is the first time I refer to the operating system popularly known as Linux, I pause to explain why I am not using the more technically correct name GNU/Linux. Richard Stallman is a key founder of the Free Software Foundation and is considered the primary progenitor of "free" software. McGowan, *supra* note 4, at 260–61. Stallman's term for what I and others call open-source software is "free" software, where "free" means freedom to use and modify, not necessarily zero cost to acquire. RPS, GNU, *The Free Software Definition*, *at* http://www.gnu.org/philosophy/free-sw.html (last visited Feb. 4, 2004). There is a debate in the open-source community about which label is best. *See* David Wheeler, *Why Open Source Software/Free Software (OSS/FS)? Look at the Numbers!*, § A.1.3, *at* http://www.dwheeler.com/oss_fs_why.html (revised Dec. 31, 2003) (summarizing positions on each side of debate, where proponents of term "free" software, including Stallman, find moral or social reasons why software should be supplied under what I call the open-source approach, and where proponents of term "open-source" advocate that practical considerations substantiate open-source approach, such as superior capabilities, cost or quality of software, and noting that "to some people, the *connotations* and motives are different between the two terms").

But I put that debate aside to discuss the naming issue. Stallman advocates that Linux is properly called GNU/Linux. To understand his proposition, one must realize that what is popularly called the Linux operating system is an aggregated and integrated set of software components that interoperate. The label GNU/Linux provides attribution for more than the kernel of the operating system developed by Linus Torvalds. The GNU part recognizes that many significant components of the operating system are from Stallman's Free Software Foundation and originated before Torvalds began developing his kernel. Benkler, *supra* note 4, at 371 n.3. Some would probably argue that I help perpetuate the problem by using the popular name, but I do so for the reader's convenience while recognizing that "the complete GNU/Linux operating system is what everyone has in mind . . . ." *Id.*

[13]Patrick K. Bobko, *Linux and General Public Licenses: Can Copyright Keep "Open Source" Software Free?*, 28 AIPLA Q.J. 81, 85–86 & n.20 (2000); Peter Brown & Lauren McCollester, *Should We Kill the Dinosaurs or Will They Die of Natural Causes?*, 9 CORNELL J.L. & PUB. POL'Y 223, 233 (1999); Maher, *supra* note 5, at 684–86.

[14]Peter G. Neumann, *Robust Open-Source Software*, 41 COMMUNICATIONS OF THE ACM No. 2, at 128, 128 (Feb. 1998).

[15]McGowan, *supra* note 4, at 242–43.

the open-source approach with entitlements that the civil law tradition of European-based legal systems traditionally provide to authors and artists. Doing so illuminates legal and policy rationales that suggest three claims for which I argue. My third claim is the most provocative among the three: that the legacy of the civil law right of integrity, which allows the author to preserve the state of her work against certain incursions, suggests that open-source software would benefit from a statutorily enacted right of "Collaborative Integrity" to preserve the open-source, royalty-free, state of the software, i.e., to preserve the software's facility for collaboration.

European legal systems endow authors and artists with the right to control aspects of their work even after they have sold the work.[16] For example, a sculptor in France may sell a statue, but retains a "right of integrity" in the statue to object to certain distortions, mutilations, or other modifications.[17] If the buyer mutilated the statue by painting it purple, the sculptor may have legal recourse to remedy the mutilation.[18] In essence, the European right of integrity allows the sculptor to govern, in certain respects, the view that the work presents. Throughout this Article, I use both European and international examples of the civil law's traditional right of integrity because the modern manifestation of these rights, both outside of Europe and in international treaties, trace their development to eighteenth century Europe.[19]

Developers of open-source software distribute source code with the software under a generally applicable license. The license provides conditional permission to use copyright protected material. In the set of conditions I call the open-source approach, the license would require, as a condition of use, that recipients redistributing the software also redistribute the source code. More importantly, if the recipients modify the source code, they must make the modifications available to others if they redistribute the software. Finally, recipients who redistribute (with or without modifications) must impose the

---

[16]JEREMY J. PHILLIPS ET AL., WHALE ON COPYRIGHT 15–16 (5th ed. 1997).

[17]Thomas F. Cotter, *Pragmatism, Economics, and the Droit Moral*, 76 N.C. L. REV. 1, 5 (1997); Henry Hansmann & Marina Santilli, *Authors' and Artists' Moral Rights: A Comparative Legal and Economic Analysis*, 26 J. LEGAL STUD. 95, 99–100 (1997); Susan P. Liemer, *Understanding Artists' Moral Rights: A Primer*, 7 B.U. PUB. INT. L.J. 41, 41–42, 50–51 (1998); Neil Netanel, *Alienability Restrictions and the Enhancement of Author Autonomy in United States and Continental Copyright Law*, 12 CARDOZO ARTS & ENT. L.J. 1, 24 (1994).

[18]Neil Netanel, *Copyright Alienability Restrictions and the Enhancement of Author Autonomy: A Normative Evaluation*, 24 RUTGERS L.J. 347, 387–88 (1993).

[19]Beyond my choice to focus on the civil law tradition of moral rights, and specifically the rights of attribution and integrity, is a more general question: why compare open-source software licensing to such rights, rather than compare it to other regimes that might have collaborative influences? My reasons are pragmatic. Noticing the parallels between the right of integrity and open-source software licensing prompted my interest in the analysis. While I do not argue in this Article that my proposed right of Collaborative Integrity is an optimal system of protection, explaining these parallels is instructive. My thanks to Mike Madison for suggesting that I clarify this aspect of my comparison.

same terms on their licensees. In essence, under this approach, open-source programmers govern how the "work" (the software) will be viewed by future users and developers. It must be viewable at least as source code.

There is a parallel impression comparing the civil law right of integrity with the open-source approach: control over the view that the work presents. This parallel suggests, in some sense, a "right of integrity" in the open-source approach. The author's or sculptor's interests are protected by the right of integrity, whereas the open-source programmer's interests are protected by the generally applicable license.

This comparison forms the basis for my three claims. First, that the open-source approach can be better understood in light of the comparative assessment. Second, that some non-U.S. legal systems may already provide a degree of latent alternative protection for the open-source approach because these systems provide moral rights in software to some degree. The hypothesis is that violation of the open-source licensing permissions can be thought to violate the programmer-authors' moral rights in these non-U.S. legal systems. And, third, that the legal community should evaluate an altered approach to protecting open-source software. Specifically, I sketch an altered model suggested by the moral rights entitlement that the European legal systems provide to authors and artists,[20] but modified for the collaborative nature of software development. I call the alternative model "Collaborative Integrity" for open-source software.

Collaborative Integrity, which would be best implemented under a statutory public law mechanism, proposes to blend these notions and extend them to account for the multiple creators contributing cooperatively to develop software. I will explore in this Article the legal and policy arguments for and against such an approach. Open source is a new phenomenon, arising along with, and at the speed of, the Internet. Our understanding of this phenomenon is strained to keep pace. Considering it in light of traditional civil law moral rights granted to authors and artists enhances our understanding of the open-source movement. This is especially important as the open-source approach grows internationally. Copyright and licensing law, the basis of the open-source approach, is different in some ways in non-U.S. legal systems,[21] and therefore, the open-source approach may have differing implications in these systems. In developing my three claims, this Article seeks to help the law

---

[20] *See* PAUL GOLDSTEIN, INTERNATIONAL COPYRIGHT: PRINCIPLES, LAW AND PRACTICE viii–ix, 8–10 (2001) (using French and German law as primary examples, describing European authors' rights tradition and how it differs between French and German law, and arguing that natural rights philosophy thought to underlie authors' rights and separate it from economic and incentive-based theories of copyright is less acute in formation of those rights than commonly espoused).

[21] 15 JOSEF DREXL, WHAT IS PROTECTED IN A COMPUTER PROGRAM? COPYRIGHT PROTECTION IN THE UNITED STATES AND EUROPE 2–3 (Friedrich-Karl Beier & Gerhard Schricker eds., 1994).

respond to the open-source movement by contributing to the expanding scholarship about its implications, effects, and promises.

Part II of this Article begins by discussing the significance of source code. Software takes different forms at different stages in the development process. Source code is the most important collaborative form. Thus, I review its relationship to object code within a framework describing the computing process. In open-source software projects, the collaboration occurs by functionally intermingling, layering, and linking source code texts. Often, but not always, multiple programmers contribute to each text, sometimes concurrently, but also sequentially. Important to this process is the legal protection that attaches to source code, which Part II summarizes. Open and available source code is a predicate for collaboration, and open-source software uses intellectual property protection in a unique way to enable the collaboration.[22]

Part III reviews open-source software. The open-source movement began as an ideology cleverly implemented through copyright to make the source code for software available and viewable.[23] This, along with other evolving forces in computing, fostered software development under the open-source model.[24] Products grew out of these development projects. As the products attracted users, especially within the burgeoning Internet infrastructure, the open-source approach took on a life of its own, attracting investment, entrepreneurs, and ultimately support from many of the largest computing companies in existence. Along the way, legal, business, computer science, and economic commentators began to pay attention, resulting in an increasing scholarship devoted to the open-source movement.

---

[22]In very broad and brief terms, the premise is that software development is a creative activity that benefits from lessening the effective protection of the works, or preserving a realm of use for others, rather than successively endowing the work with control-enabling rights. *See* Julie E. Cohen, *Lochner in Cyberspace: The New Economic Orthodoxy of "Rights Management"*, 97 MICH. L. REV. 462, 466, 530 n.258 (1998) (arguing that due to "the special nature of creative and informational works and of creative and intellectual progress, there is substantial reason to believe that a limited-ownership regime is better suited to furthering these goals," and noting that the open-source operating-system Linux, as well as the GNU operating-system project, are examples of this effect).

[23]McGowan, *supra* note 4, at 253–56; *see also* ROBERT BOND, E-LICENSES AND SOFTWARE CONTRACTS: LAW, PRACTICE AND PRECEDENTS 24–25 (2000) (describing two significant open-source activist organizations and their techniques to implement or define open-source approach, including the requirement that source code be made available with software).

[24]*See* Yutaka Yamauchi et al., *Collaboration with Lean Media: How Open-Source Software Succeeds*, 2000 ACM CONF. ON COMPUTER SUPPORTED COOPERATIVE WORK 329, 329 (describing that although software development has traditionally been "a coordination-intensive process," recent interest and advances in computer supported cooperative work facilitated geographically distributed nature of much open-source software development).

Following the open-source software discussion, Part IV outlines the traditional civil law *droit moral*, or moral right,[25] focusing on the rights of attribution and integrity. Traditionally, authors' rights in the civil law tradition include (1) the right to publish the work; (2) the right to have the author's name, and no other name, attributed to the work; (3) the right to object to impaired integrity of the work, that is, mutilations, modifications, or distortions of the work detrimental to the author's or artist's honor or reputation; and, most obscurely, (4) the right to withdraw the work in certain situations on equitable terms.[26] Among these, the right of attribution and the right of integrity are most relevant.[27] They give authors control over aspects of their work in ways analogous to open-source software's governance of source code. My thesis, in part, is to compare and contrast the control offered by moral rights with open-source software control, arguing that a better understanding of the open-source approach will result. Artists' and authors' moral rights vary by jurisdiction, thus I resort to the Berne Convention's provisions when a

[25]The phrase 'moral right' is a translation of the French phrase *droit moral*, and as such there is the possibility of lost nuance in meaning. However, most English-language commentators use the phrase "moral rights," and I follow that practice. Roberta Rosenthal Kwall, *Copyright and the Moral Right: Is An American Marriage Possible?*, 38 VAND. L. REV. 1, 3 n.6 (1985).

[26]Dane S. Ciolino, *Moral Rights and Real Obligations: A Property-Law Framework for the Protection of Authors' Moral Rights*, 69 TUL. L. REV. 935, 937 (1995); Hansmann & Santilli, *supra* note 17, at 95–96; James M. Treece, *American Law Analogues of the Author's Moral Right*, 16 AM. J. COMP. L. 487, 487, 494, 499–500 (1968–69).

[27]André Françon, *Protection of Artists' Moral Rights and the Internet*, 5 PERSPECTIVES ON INTELLECTUAL PROPERTY: THE INTERNET AND AUTHORS' RIGHTS 73, 76 (Frederic Pollaud-Dulian ed., 1999) (arguing that rights of attribution and integrity are most applicable to works disseminated over the Internet).

common expression of these rights is necessary.[28] The scholarship and commentary on author's rights is voluminous, and in this Article I do not chronicle its breadth or scope. Rather, I sample from the literature to set the stage for Part V.

Part V compares and contrasts the rights of attribution and integrity with open-source software. Specifically, the approach aligns these rights with the most common provisions of typical open-source licenses, demonstrating how each in their own way confers a degree of control over the respective works. Authors' moral rights are granted directly by statute. The right of integrity provides the author or artist some control over the view a work presents—it cannot have its integrity impaired such that those viewing the work see a mutilation, distortion, or modification. The open-source approach, to similar effect, by conditionally permitting use of a copyrighted work, enables the original and subsequent programmers to ensure that the source code is viewable. Any users, licensees, or redistributors of the open-source code risk a copyright infringement action if they do not follow the conditions permitting the open-source use.

Inspired by the civil law tradition granting authors' and artists' a right to integrity in their works,[29] Part V posits an altered framework for the open-source software approach. With the right of integrity, authors and artists can

---

[28]Berne Convention for the Protection of Literary and Artistic Works, Sept. 9, 1979, S. TREATY DOC. NO. 99-27, at art. 6*bis* (1986), Hein's No. KAV 2245, at 41, *available at* http://www.wipo.int/clea/docs/en/wo/wo001en.htm (last visited Feb. 6, 2004) [hereinafter Berne Convention]. The Berne Convention has grown in importance for international copyright as a result of the establishment of the World Trade Organization ("WTO") and its annex treaty, Trade-Related Aspects of Intellectual Property Rights ("TRIPS"), because TRIPS implemented a substantive minimum set of copyright requirements for countries that join the WTO. *See* Neil W. Netanel, *The Next Round: The Impact of the WIPO Copyright Treaty on TRIPS Dispute Settlement*, 37 VA. J. INT'L L. 441, 451 (1997) ("TRIPS effectively incorporates by reference all of the Berne Convention's substantive provisions, except as those provisions may concern the moral rights of attribution or integrity conferred under article 6*bis* of Berne."); J.H. Reichman, *Compliance with the TRIPS Agreement: Introduction to a Scholarly Debate*, 29 VAND. J. TRANSNAT'L L. 363, 366–67, 382 (1996) (noting that TRIPS establishes minimum standards for many areas of intellectual property, including copyright). TRIPS, however, did not require WTO signatory countries to provide the moral rights of attribution or integrity in local law. Agreement on Trade-Related Aspects of Intellectual Property Rights, Apr. 15, 1994, Marrakesh Agreement Establishing the World Trade Organization, Annex 1C, at art. 9, LEGAL INSTRUMENTS—RESULTS OF THE URUGUAY ROUND vol. 31, 33 I.L.M. 81 (1994), *reprinted in* THE RESULTS OF THE URUGUAY ROUND OF MULTILATERAL TRADE NEGOTIATIONS: THE LEGAL TEXTS 6–19, 365–403 (GATT Secretariat ed. 1994), *available at* http://www.wto.org/english/docs_e/legal_e/27-trips_01_e.htm (last visited Feb. 6, 2004); *see also* Netanel, *supra*, at 451. Berne's expression of moral rights is minimalist, it specifies only two of the traditional moral rights. Adolf Dietz, *The Moral Right of the Author: Moral Rights and the Civil Law Countries*, 19 COLUM.-VLA J.L. & ARTS 199, 203 (1995).

[29]*See* Calvin D. Peeler, *From the Providence of Kings to Copyrighted Things (and French Moral Rights)*, 9 IND. INT'L & COMP. L. REV. 423, 438–39, 448–49 (1999) (describing the right of integrity's early development in France).

control their works by objecting to certain modifications. This is a prohibition on certain types of modifications. In another sense, however, this control preserves essential characteristics of the work. Open-source programmers similarly seek to preserve essential characteristics of their software.[30] They control their code, by making it viewable, to preserve the collaborative opportunity, and sometimes collaborative necessity, for others to leverage their work.

Collaborative necessity stems from the functional nature of software. Functional collaboration with components that must interoperate distinguishes most software development from most authorship and artistic endeavor.[31] This collaboration is an essential characteristic of much software, whether developed under the open-source model or the traditional software development model.[32] Collaboration is an overt goal of the open-source approach because it emphasizes viewable source code. As such, recognizing that the functional nature of software and, in particular, the collaborative nature of open-source software, means that imposing the opportunity to modify open-source code serves parallel interests, as the right of integrity serves for literary and artistic works.

---

[30]At first blush, my comparison may seem to indicate more difference than similarity. The right of integrity specifically indicates that the author or artist can object to modifications. This implies a degree of control over modifications. On the other hand, the open-source approach demands that the opportunity for modification be preserved by ensuring source code access. My argument is, however, that the approaches are similar for the most important aspects of each type of work.

[31]There are counterexamples to this assertion, such as a movie, which is a team project with collaboration to some degree, but the traditional art and literary works are mostly stand-alone products. BROOKS, *supra* note 3, at 255–56. This is particularly true under copyright's traditional conceptions of authorship. *See* Peter Jaszi, *Toward a Theory of Copyright: The Metamorphoses of "Authorship,"* 1991 DUKE L.J. 455, 472 ("The 'authorship' concept, with its roots in notions of individual self-proprietorship, provided the rationale for thinking of literary productions as personal property with various associated attributes including alienability."); Jessica Litman, *The Public Domain*, 39 EMORY L.J. 965, 967 (1990) (noting establishment view that "copyright's paradigm of authorship credits the author with bringing something wholly new into the world [and challenging this paradigm because] it sometimes fails to account for the raw material that all authors use").

[32]Traditional software development employs classic group organizational methods where managers subdivide work and delegate tasks to a hierarchy of employees to implement. *See* BROOKS, *supra* note 3, at 32–33 (analogizing programming team to surgical team); *see also infra* Part III.A.1.(a).

The parallel nature of these interests is demonstrated in several contexts. First, both systems call forth personality theories for rights in intangibles.[33] Authors and artists invest their personality in their works. Open-source programmers share this characteristic. Many programmers develop or contribute to open-source projects as a hobby or pastime, or for ideological reasons separate and apart from their gainful employment. Thus, many open-source programmers make a personal investment in the code. Second, in the civil law system, moral rights exist to some degree separate and apart from the economic "copyright" rights in the work.[34] The open-source approach has accomplished something similar by using the license to impose additional conditions beyond copyright protection. The separateness is less acute because the license depends on the underlying copyright protection. The open-source license, however, has unique goals compared to copyright and implies a separate ideology. Demonstrating this is the moniker "copyleft"—a play on words meant to express that the open-source goal, making the work mostly available, is opposite an ex-post or pejorative view of copyright's function: generally protecting and prohibiting use of the work by others, while perhaps licensing some narrow use.[35] Third, both systems have organizational and institutional designs and effects influenced by the rights granted to the authors and artists, and programmers, respectively. Authors' and artists' moral rights may affect their interactions with other groups, such as publishers or

[33]*See* Cotter, *supra* note 17, at 7–8 (reviewing philosophical antecedents of European civil law moral rights and noting that it sprang from personality theory notion that "the thing possessed comes to embody the owner's personality," and also discussing contrast between personality theory as basis for rights with instrumentalist theory of providing rights to create incentives for certain outcomes); Margaret Jane Radin, *Property and Personhood*, 34 STAN. L. REV. 957, 1013 n.202 (1982) (noting that moral rights are manifestation of property rights rooted in theories of personality, and that moral rights go "beyond copyright, which protects only against economic exploitation of one's work by others, to give the artist the right to prevent owners of her work from altering or destroying it"); *but see* GOLDSTEIN, *supra* note 20, at 8–10 (arguing that philosophy thought to underlie authors' rights and separate it from economic and incentive-based theories of copyright is less acute in formation of those rights than commonly espoused).

[34]Hansmann & Santilli, *supra* note 17, at 95.

[35]DONALD K. ROSENBERG, OPEN SOURCE: THE UNAUTHORIZED WHITE PAPERS 90–91 (2000); Heffan, *supra* note 5, at 1491. Under an ex-ante or more favorable view of copyright's goal, to create economic incentives for creation and dissemination of works, copyright shares similarities to copylefting or open sourcing works: both seek wide dissemination. The difference, of course, is the mechanism of dissemination. Copyright relies on commoditization and the pricing mechanism to obtain dissemination, while open-source software relies on self-organizing critical mass for developers and users.

distributors.[36] Open-source programmers, operating under an obligation to make source code available upon redistribution, have less need for formal organizational structures.[37] Their collaboration, while necessarily coordinated to a certain extent, can be, and often is, loose to a degree heretofore unprecedented in software development.

Although originally developed as judicial doctrines, moral rights in the civil law systems have served the interests of authors and artists in later years as statutory protections.[38] To the extent these interests parallel those of open-source programmers and the open-source movement in general, statutory protection suggests itself for some aspects of the open-source approach. In response to this suggestion, I argue for such protection under a right of Collaborative Integrity as my third claim in this Article. The various open-source licenses permit use under a number of differing conditions. Those conditions that express the collaborative essence of the open-source approach could be beneficially infused with statutory authority. Doing so should further heighten the incentives for the creation of and contribution to open-source software. It could facilitate evolving forms of open-source project organization and management. Alternatively, it could supplement the enforcement power of open-source licenses that seek to attach additional conditions on open-source use. At a policy level, it would express an approval of the volunteerism inherent in the open-source movement. Thus, in Part V, I elucidate the collaborative essence of the open-source approach, which I label the Collaborative Integrity of open-source software.

Part VI then concludes by emphasizing the implications of Collaborative Integrity. It recounts the dramatic progression of open-source software from an ideology, to an Internet-niche technology, to a major competitive force in the most important of software arenas: computer operating systems and the Internet. It stresses the parallels between the open-source approach and authors' and artists' rights to attribution and integrity. These parallels will help us better understand the open-source approach, suggest alternative grounds to enforce it in jurisdictions where moral rights apply to software, and imply a need to evaluate alternative mechanisms to enable and support the open-source approach. Recognizing the Collaborative Integrity of open-source software

---

[36]*See* Jane C. Ginsburg, *A Tale of Two Copyrights: Literary Property in Revolutionary France and America*, 64 TUL. L. REV. 991, 1011–13 (1990) (suggesting that traditional accounts of initial establishment of authors' and artists' moral rights in eighteenth-century France overstate degree to which concerns for authors and artists animated discussion because "generally the most vociferous advocates for authors' rights were not authors, but their publishers, . . . [a]rguments for copyright therefore evoked images of guild self-interest in a period of increasing anticorporatism" (citation omitted)).

[37]Benkler, *supra* note 4, at 377–78, 413, 435–36.

[38]Cotter, *supra* note 17, at 5.

provides the opportunity to give it, and the unique approach upon which it is based, greater effect and efficacy.

II. THE SIGNIFICANCE OF SOURCE CODE

Open-source software has a recent history spanning about the last decade and a half. Its rise correlates to the rise of the Internet. Its history springs from rebellion against certain traditional software development practices. As a result, understanding its origins requires some understanding of what came before it, what competes with it today, and why its holy grail is source code availability.

The next Section explains, partly by metaphor, software development basics and the importance of source code to the computing process. This enables my later discussion of traditional software development contrasted with open-source software.

A. Software and Source Code

When developing software, one arranges a composite of instructions, data, and interfaces in a sequence and hierarchy that will produce a particular desired computing outcome.[39] This instructional composite bears the label "software," or "computer program." The U.S. copyright statute's definition reflects software's nature: "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."[40] The definition reflects three important concepts, elaborated below.

1. The Three Elements of Computing: Instructional Composites, Operating Systems, and Computing Results

Three concepts inhere in the functional nature of a computer program. I label these concepts the three elements of computing. Each element is present

---

[39]HAROLD ABELSON ET AL., STRUCTURE AND INTERPRETATION OF COMPUTER PROGRAMS xii–xiii, 1, 4–5, 79–80, 217–18 (2d ed. 1996) [hereinafter ABELSON ET AL., COMPUTER PROGRAMS] ("Computational processes are abstract beings that inhabit computers. As they evolve, processes manipulate other abstract things called *data*. The evolution of a process is directed by a pattern of rules called a *program*. People create programs to direct processes."); ANTHONY LAWRENCE CLAPES, SOFTWARS: THE LEGAL BATTLES FOR CONTROL OF THE GLOBAL SOFTWARE INDUSTRY 12–16 (1993) [hereinafter CLAPES, SOFTWARS].

[40]17 U.S.C. § 101 (2000).

in the copyright statute's definition of a computer program.[41] First, the statements or instructions, which are a hierarchical composite of interacting commands, procedures, structures, data, variables, and interfaces. A programmer composes the composite to make the program. Second, the computer, or more properly, the computer's operating system. The operating system is the means or the agent by which the instructions are carried out. Third, the certain result, that is, the desired computing output.[42]

The instructional composite is the lynchpin of all three computing elements. It defines what the computer will do. It is a necessary, but not sufficient, predicate to a successful computing result. It is what many people are referring to, in part, when they use the term "source code."[43] The instructional composite, however, takes different forms at different stages in the software development process. These variations in form produce the crux of one problem at which open-source software is aimed: a nonhuman readable form of the instructional composite, often called the "object code," is the only instructional composite available with most traditional software.[44] To the progenitors of the open-source movement, this reduced the value of the software and offended values important to that community of technologists. They preferred that software also deliver the human-readable form of the

---

[41]*See also* William F. Patry, *Copyright and Computer Programs: It's All in the Definition*, 14 CARDOZO ARTS & ENT. L.J. 1, 17–18, 22–24, 30–32 (1996) (discussing definition of computer program, and history of definition, in arguing, among other points, that copyright protection for computer programs should pay greater attention to definition in section 101).

[42]To give an example touching on all three elements of computing, consider the spell-checking component of a word-processing software package. We can conceive of the sequence and process that the spell-checking program undertakes. It models how we would check the spelling in a document ourselves. We would read through the document, scanning for words that were misspelled. The scanning step requires further elaboration. Each time we read a word, we compare it to the list of words we have stored in our memory. If the word is not recognized, we try to correct it by comparative contextual estimation. That is, we find known words similar to the unknown word and determine whether they fit the context. The spell-checking program operates similarly.

My thumbnail description of how we would spell check is the starting point for writing one possible instructional composite directing a computer program that checks spelling. The second element for this example is the computer's operating system, which must be able to read or interpret the spell-checking instructional composite (or some transformed version of it) and perform the third element, actually checking the spelling in a document.

[43]*See* Bobko, *supra* note 5, at 73 (noting, in discussion of nonliteral copyright infringement for structure, sequence, and organization of software, importance of items such as data structures defined by source code).

[44]With today's computing technology, the object code form of the instructional composite is stored in computing media, whether it is a disk drive or memory, encoded as ones and zeros. While the object code is typically thought to be nonreadable by humans, some extremely gifted humans can read the object code form directly. LAWRENCE LESSIG, CODE AND OTHER LAWS OF CYBERSPACE 103 (1999) (noting that only geniuses and computers can read object code).

instructional composite.[45] This form is the computer program as implemented in a specific computer programming language. Other programmers cannot readily modify or learn from software that lacks its human-readable instructional composite. In other words, without the source code, collaborative possibility is diminished.

To illustrate, I next develop an analogy comparing the instructional composite's variations in form with translating human languages, such as translating from German to Japanese. This analogy highlights the importance of making the source code available with the software. To emphasize this importance, I develop the analogy in the Subsection below using a metaphor that compares the computing process to the cooking process.

2. The "Software as Recipe" Metaphor for the Three Elements

Computing's three elements are the instructional composite, the operating system, and the computing result. Cooking has three analogous elements: the recipe, the cook, and the dish.

Assume that Gerhard is a person who speaks and writes only German. He creates a wonderful recipe for Bavarian-style coriander chicken. Gerhard wants his cousin Francis, who lives in Japan, to experience Bavarian-style coriander chicken. Francis, however, does not cook, although his kitchen is fully equipped to make the recipe. Francis employs a cook who reads, writes, and speaks only Japanese. The cook is fully skilled and capable to make Bavarian-style coriander chicken if the recipe is presented to the cook written in Japanese. The obvious solution is to translate the recipe from German into Japanese.

In this analogy, the German recipe is like the source code of a computer program. Alternatively put, the German recipe is the German-readable form of the instructional composite for making Bavarian-style coriander chicken. The cook and kitchen combination is like the operating system and computer combination. The cook uses the "hardware" of the kitchen to implement the recipe for the desired dish. The operating system directs the computer hardware to produce the computing result.[46] The translated Japanese version of the recipe is like the object code that a computer directly executes. Alternatively put, the Japanese version is the computer-readable form of the instructional composite.

Computer programs are written in languages analogous to German (or English, French, etc.), with vocabulary, acceptable syntax, grammar, and many other features characteristic of traditional human languages. Computer

---

[45]Richard Stallman, *The GNU Operating System and the Free Software Movement*, *in* OPENSOURCES, *supra* note 2, at 53, 53–55.

[46]RANDAL E. BRYANT & DAVID R. O'HALLARON, COMPUTER SYSTEMS: A PROGRAMMER'S PERSPECTIVE 13–14 (2003).

program languages go by sometimes-funny names, such as "C" or "C++" or "FORTRAN" or "Pascal" or "Java." The allowed statements and syntax for these languages use, mostly, English words. As a result, a non-programmer who reviewed source code in the "C" programming language would recognize some words, such as "main" or "for" or "return." These words, as they appear in a "C" program, have meanings different from their use in written or spoken English. To one who knows the "C" programming language, however, they have precise meanings related to directing the operating system to direct the hardware to undertake specific operations in particular ways.[47] In similar fashion, Gerhard's recipe instructs one who can read German what operations to perform to make Bavarian-style coriander chicken.

Like Gerhard and Francis' language mismatch, software development technology has a language mismatch. As with the cook who can only read Japanese, computers directly read and execute "object code" instructions. The program written in the "C" programming language must be translated into "object code" that the computer can read and execute.[48] Alternatively put, the instructional composite expressed in "C" must be translated into an instructional composite that the computer can directly read and execute.

Unlike the human language difference, which is a historical and cultural artifact, the difference in computing instructional composites is from conscious design. The "object code" that the computer can directly read and execute is too awkward and unwieldy for human computer programmers to productively read and write. This is due to at least two causes. First, the "object code" is typically encoded as a series of "operational codes" that correspond to the basic operations that the computer can perform, typically in the range of a few hundred.[49] Although all information in current computers is stored as ones and zeros, much of it is encoded such that when one views the information through standard viewers (which are themselves software programs), human-readable characters result.[50] This is not the case with "object code"—it is only encoded to be intelligible by the computer's hardware. Specifically, the "object code"

---

[47]ABELSON ET AL., COMPUTER PROGRAMS, *supra* note 39, at xvii–xviii ("[P]rograms must be written for people to read, and incidentally for machines to execute.").

[48]BRYANT & O'HALLARON, *supra* note 46, at 4–5; Rachiver, *supra* note 5, at *6–*8.

[49]BRYANT & O'HALLARON, *supra* note 46, at 263–65.

[50]The encoding scheme used to store and retrieve/decode the information depends on a variety of factors, including the type of data stored and the age of the computing technology at issue. For example, a still-pervasive but relatively old encoding scheme is the American Standard Code for Information Interchange, or "ASCII" code. In the ASCII code, unique 8-bit binary numbers are assigned to the English alphabet and other common characters one finds on a computer keyboard. MICHAEL D. SCOTT, INTERNET AND TECHNOLOGY LAW DESK REFERENCE 42 (2003) [hereinafter SCOTT, I&T LAW REF.]; ASCII Table, ASCII Table and Description, *at* http://www.asciitable.com (last visited Feb. 7, 2004). The original ASCII code had 128 such associations. For example, the binary code for "A" is 01000001, but the code for "a" is 01100001. The same series of 8 ones or zeros, however, could be interrupted under a different encoding scheme to mean something completely different.

expresses the instructional composite in terms of operations or instructions that the computer's hardware can perform.[51] Returning to the cooking analogy, the Japanese recipe is the "object code." If the Japanese recipe says "set temperature to 177 degrees Celsius," the cook (i.e., the operating system) reads the instruction and commands the kitchen hardware (i.e., the computer) to carry out this instruction using its hardware, specifically, the thermostat on the oven.

The second reason that "object code" is unproductive and unwieldy for writing programs is that the available operations in most computer hardware are too limited and basic. They are too "low-level." Writing programs using these operations to perform standard data-processing operations, such as sorting data, or running the same routine on different data again and again, is cumbersome and costly.[52] That is why programming languages such as "C" and "Pascal" are called high-level programming languages—they allow economy of expression by hiding the details for common operations.[53]

Specifying these additional details takes time and makes the programming work less interesting, thus programmers prefer to program in high-level source code. Hiding the details is work that can be automated—meaning that a computer program can assist with the software development process. The ultimate end goal is to create executable object code. A special computer program, called a compiler, helps the code get from high-level human-readable source code to object code. This computer program has a special role in developing software. It translates, just like the translator who helped Gerhard provide the Bavarian-style coriander chicken recipe to Francis.

3. Source Code: The Link Between the First and Second Element

Programmers are much more productive working in source code compared to object code. It is better if they can perform basic operations using language statements like "bake for twenty minutes at 170 degrees Celsius," rather than specifying all the step-by-step operational details. As a result, software development technology emphasizes high-level instructional

---

[51]BRYANT & O'HALLARON, *supra* note 46, at 124 (noting that "in a high-level language such as C," programmer need not "specify exactly how the program manages memory and the low-level instructions the program uses to carry out the computation").

[52]*See* BROOKS, *supra* note 3, at 186.

[53]To illustrate, consider the cooking analogy, where the high-level language statement is "bake the chicken for twenty minutes at 177 degrees Celsius." The low-level operations would typically be much more voluminous. They might say: (1) unlatch the oven door, (2) open the oven door, (3) check that the oven is empty, (4) insert the chicken, (5) close the oven door, (6) latch the oven door, (7) set the timer to go off in 1200 seconds, (8) set the heat to 177 degrees Celsius, (9) start the oven, (10) wait for the timer, (11) when the timer goes off, turn off the oven, (12) reset the heat to 0 degrees Celsius, (13) unlatch the door, (14) open the door, and (15) remove the chicken.

composites where the language statements and structures can do more work. To support this approach, the technology turned to automated translators, called compilers. Compilers generate low-level object code from the high-level source code.[54] They change the form of the instructional composite in much the same way that the human translator converts Gerhard's Bavarian-style coriander chicken recipe from German to Japanese.

The copyright statute's definition of "computer program" implicitly recognizes the compilation step that changes the instructional composite from source code to object code. It says that 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." One way to read this definition is that the "direct" instructions are object code, but the "indirect" instructions are source code.[55]

The source code is the work studio of the programmer. It must be so; the object code is too unwieldy. The source code is the environment in which the programmer solves computing problems and codifies the solutions. This environment is aware of all three elements of computing. The desired computing result anchors the coding process. The target operating system and hardware influence the process as well. Just as a blueprint shows how a house is built, or a recipe for coriander chicken illustrates how one makes the delectable dish, a computer program's source code conveys much information to other programmers. It shows the solution the programmer implemented for the computing task at hand. It may show the programmer's technological elegance, personal wit, determination, charm, genius, or pedestrian skills.

The source code includes more than just statements that command the operating system to do something. It typically contains comments, taken in the ordinary sense of the word. Through comments, programmers insert information into the computer program that does not become part of the object code.[56] Even though not used in the object code, comments can play an

---

[54]BRYANT & O'HALLARON, *supra* note 46, at 4–6.

[55]1 MELVILLE B. NIMMER & DAVID NIMMER, NIMMER ON COPYRIGHT, § 2.04[C] (2003) ("A computer program by definition is 'a set of statements or instructions to be used directly or indirectly in a computer.' A source code program is used 'indirectly,' and an object code program is used 'directly.'" (quoting 17 U.S.C. § 101 (2000))).

[56]The compiler (a special computer program) filters out comments when it transforms the source code instructional composite into the object code expression of that composite.

important role in making the source code valuable.[57] One can often glean the history associated with developing a computer program 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. Sometimes even legal notices, such as a copyright or licensing notice, are placed in source code comments. In sum, comments record the program's history and thus can facilitate collaboration and reuse of the code.

Ultimately, the development process requires that product from the source code work studio be transformed into object code by the compiler so that the operating system can read and execute it using the computer's hardware.[58] The compiler is itself a computer program with the special task of translating the source code to object code. The compiler unifies the three elements of computing by taking the "indirect" instructions of the source code instructional composite and preparing an object code instructional composite for the operating system.[59] The German recipe is translated into Japanese, and the cook and kitchen carry out the recipe, just as the operating system and computer hardware carry out the program.

---

[57]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. Some programmers prefer to not rely on comments to document the source code. Their theory is that other techniques do the job better, such as choosing well-named variables in source code. ABELSON ET AL., COMPUTER PROGRAMS, *supra* note 39, at 124 n.21; *see also* BROOKS, *supra* note 3, at 172–74. Programming languages allow the programmer to choose names for units or sets of information. These names are called variable names. They may be as simple as "AutoPrice" to hold a single price for a single car; or complex as in "AutoPrices[25][35]" to hold a matrix or table of prices for up to twenty-five cars in thirty-five separate countries; or even much more complex and intricate. 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 structure and hard to understand variable names readily understandable.

[58]*See* McRoberts Software, Inc. v. Media 100, Inc., 329 F.3d 557, 562 (7th Cir. 2003) (discussing defendant's translation of video-editing character-generation software which ran only on Macintosh computer and operating system to source code instructional composite that would compile and run on Windows-based computers, which process plaintiff described as "akin to translating English to Chinese").

[59]As one might expect, the analogy of program compilation to human language translation is a poor fit in a variety of minute technical aspects. For my purposes, however, the analogy is a good fit because it emphasizes the importance of access to the source code. Similarly, my direct descriptions of the software development process and the compilation step are necessarily quite broad. They do not reveal the rich variety of technologies and approaches that could fall under my lay description of the compilation step. Again, however, the purpose is met by emphasizing that supply of only compiled object code is a poor substitute for the source code if one wants to achieve maximum learning from the code.

The figure below depicts the transformation of the source code within the three computing elements model.

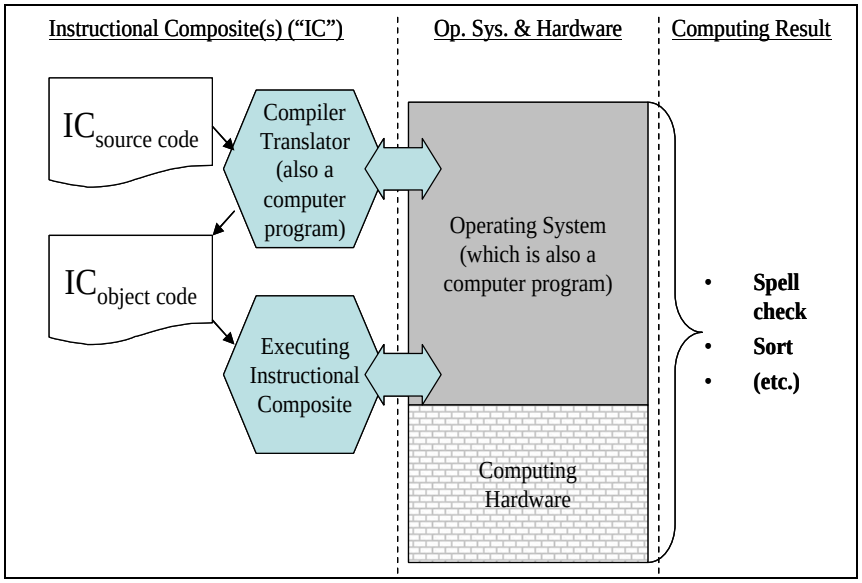

Figure 1

The Three Elements of Computing

In this figure, the shaded, six-sided objects are computer programs executing "on top of" the operating system. The black arrows indicate the progressive transformation of the instructional composite.

The foregoing discussion demonstrates the importance of source code and its role in the software development process. Source code commands the ultimate result. It is the work studio of the programmer, implementing software to deliver a result.[60] Source code is meaningful in several ways. Programmers can study the configuration and use of its language statements. They can study how the source code organizes data, interacts with its own internal components, or makes use of the operating-system functions available to it.

---

[60] ABELSON ET AL., COMPUTER PROGRAMS, *supra* note 39, at 4. The authors state that:

> a powerful programming language is more than just a means for instructing a computer to perform tasks. The language also serves as a framework within which we organize our ideas about process. Thus, when we describe a language, we should pay particular attention to the means that the language provides for combining simple ideas to form more complex ideas.

Id.

These insights are available through the direct programming language statements that resemble the operative steps in a recipe ("bake for twenty minutes at 170 degrees Celsius"), through choices made by the programmer in naming routines, procedures, and information, or through the comments that usually are present. The comments may provide hints, narrative, design reasoning, even sly humor and thoughts for improvements not yet implemented.

Source code makes the inner workings of a computer program directly observable. Without the source code, one can sometimes observe the inputs, behaviors, and outputs and glean a general sense of the program's operation. It is sometimes even possible to create a replica of the program using only a specification of the externally observable inputs, behaviors, and outputs of the target program.[61] But these techniques are rarely as effective as access to the source code when one wants to leverage the work of another programmer. Well-documented source code is the gold standard for reusable software.

Source code is at the center of technological importance in software, and as a result, it also occupies the primary historical focus for software protection. The next Section expands the source code focus to discuss its legal protection, reviewing how the traditional modes of software protection apply to software and source code. This background enables better understanding of the open-source approach, which does not "protect" in the traditional sense of intellectual property, but, rather, seeks to protect and preserve the Collaborative Integrity of software's source code.

B. Traditional Intellectual Property Protection for Software and Source Code

The four major types of intellectual property protection apply to software. Trade secret law can protect secrets embodied in or implemented through software. Copyright, because its protection attaches upon fixation of original expression, is the dominant form of software protection.[62] Patent protection for software has grown doctrinally and in practical importance since the mid-to-late 1990s.[63] It represents a current area of policy controversy, both generally and for open-source software.[64]

---

[61]*See* BRYANT & O'HALLARON, *supra* note 46, at 124–25 (discussing reverse engineering).

[62]1 RAYMOND T. NIMMER, THE LAW OF COMPUTER TECHNOLOGY § 1:1, at 1–4 (3d ed. 2003) [hereinafter NIMMER, COMPUTER TECHNOLOGY] ("Since the 1980s, copyright law has been a major form of protection and property rights for computer programs, databases, software technology and other digital works.").

[63]G. PETER ALBERT, JR., ET AL., INTELLECTUAL PROPERTY LAW IN CYBERSPACE 416–17, 420–21 (1999); McJohn, *supra* note 5, at 47 ("[T]he law on patentability of computer-related inventions has itself changed radically—from forbidding to welcoming.").

[64]Bessen, *supra* note 5, at 13.

Least relevant for this Article is trademark protection, although it certainly applies to software. In fact, one of the most successful open-source software projects, the Linux operating system, has high brand recognition for the trademarked moniker "Linux," under which various groups develop and distribute the software.

To catalog the modes of protection bearing on source code, the remainder of this Section highlights software protection under the traditional areas of intellectual property, excluding trademark law. Chronologically, trade secret law, then copyright law, and then patent law became important in relation to software. The circumstances, however, of each area's foothold over software protection are unique. Each area's rise reflects influences of the times when it took a prominent place in the panoply of regimes under which rights may attach to software.

1. Trade Secret Protection

Trade secret law fit early computing technology. The industrial organization and technological deployment of software before and into the 1970s called for trade secret protection. Computers were not ubiquitous. Large organizations were the primary users. Software was developed with languages that were, by today's standards, rather "low-level." As a result, those languages required a highly specialized and skilled artisan to deal with the computer program. More importantly, software was often distributed under tight contractual control, often with negotiated agreements. This facilitated trade secret control. The contracts could extract promises and duties in exchange for use of the software. Further, users of licensed software often never obtained a copy of the software. Instead, they used the software remotely. The code might actually run on a mainframe computer remotely located from the user's facility. This further facilitated protecting any trade secrets embodied in the source code of the software.

As computer technology and software languages evolved, however, other protection regimes supplemented trade secret law. Increasing standardization in languages and operating systems heightened the possibility that programmers could reverse engineer object code, obtaining in the process a close proxy for the original source code. Thus, the secret status of trade secrets embodied in software was increasingly in jeopardy of discovery, which would foil the trade secret protection. This and other drawbacks of trade secret protection triggered analysis of alternative protection.[65] Rights-holders, however, did not abandon trade secret protection. It is often included in form license agreements for traditional software. But trade secret protection is

[65]Nat'l Comm'n on New Technological Uses of Copyrighted Works, CONTU's Final Report and Recommendation (1979), *reprinted in* 5 COPYRIGHT, CONGRESS AND TECHNOLOGY: THE PUBLIC RECORD 1, 33–36 (Nicholas Henry ed., 1980) [hereinafter CONTU Final Report].

singularly inapplicable to open-source software. The accessible and open-source code would almost always defeat the trade secret status by disclosing the secret.

While still applicable to closed, traditional software, trade secret law is no longer the dominant mode of protection for software. Copyright protection holds that place.

2. Copyright Protection

Source code's literary nature became more pronounced as computing technology advanced. Software programming languages became more sophisticated and elegant. They began to increasingly resemble human languages. This suggested copyright protection, including protection of the source code as a literary work. The U.S. Government studied the problem and eventually issued an important report that recommended copyright protection for software.[66] In 1980, Congress amended the copyright statute to explicitly cover computer programs.[67] From that time forward, copyright provided software rights-holders a more certain scope of protection. Although copyright protection does not prohibit independent creation, the useful life of most computer programs, as with computing technology generally, is sufficiently short so as to very rarely, if ever, persist beyond the copyright term.

As with traditional literary works, copyright infringement of the holder's reproduction right for source code tends to fall into several categories: literal and nonliteral copying, and derivative works.[68] Literal copying is the primary copyright basis upon which the open-source approach depends. Also potentially important, however, are (1) nonliteral copying, and (2) infringement of the copyright holder's exclusive right to create derivative works. In cases of nonliteral copying, that is, allegations that one copied the structure, sequence, and organization of the source code, copyright protection

---

[66]Nicholas Henry, *Introduction* to 5 COPYRIGHT, CONGRESS AND TECHNOLOGY: THE PUBLIC RECORD ix (Nicholas Henry ed., 1980) (noting that twenty-one years of public debate led to Copyright Act of 1976). During the development of the Copyright Act of 1976 Congress established a National Commission on New Technological Uses of Copyrighted Works ("CONTU"), "which was active from 1975 through 1978 . . . ." *Id.* at xiii. "The report recommended treating computer programs as a form of literary work, assimilating databases to compilations under existing copyright principles, and abjuring special treatment of computer-generated works because no insurmountable problems had become apparent or were foreseeable." Arthur R. Miller, *Copyright Protection for Computer Programs, Databases, and Computer-Generated Works: Is Anything New Since CONTU?*, 106 HARV. L. REV. 977, 979 (1993) (citation omitted). *See also* Samuelson, *supra* note 2, at 670–71 (questioning CONTU's recommendation that computer program copyrights extend to machine-readable (object) code and discussing alternative solutions).

[67]NIMMER & NIMMER, *supra* note 55, § 2.04[C].

[68]NIMMER, COMPUTER TECHNOLOGY, *supra* note 62, § 1:14, at 1–41 to 1–52 ("[T]he software author's protection should be within traditional, limiting concepts in copyright law.").

is said to be "thin" because the copying analysis partitions protectable versus unprotectable elements of the program.[69]

There are, however, other ways to copy software than merely copying the source code. The source code is just one form of the instructional composite. The object code instructional composite is also protected against unauthorized copying as a computer program. Unauthorized copies of the object code violate the reproduction right—another instance of literal infringement. Copyright's power to protect software increased dramatically under the "RAM copies" doctrine, under which merely running a computer program was adjudged to be a violation of the reproduction right, and thus, an instance of literal infringement if done without permission or statutory authorization.[70] The doctrine greatly increased the number of actionable events that constitute reproduction of the copyrighted work. The doctrine says that copying digitally encoded content, such as the object code instructional composite of a computer program, from permanent storage, such as a hard drive into the memory of a computer, is a violation of the reproduction right.[71] This means that merely running the computer program is a copyright infringement because running the program necessarily entails copying the object code into the computer's memory so the computer processor can execute the instructional composite. By increasing copyright's power to control software, the RAM copies doctrine also increases the power of the open-source approach to control use of the software.

---

[69]With respect to nonliteral copying, what followed software's entry into the copyright regime was a period of uncertainty as the courts struggled to determine the scope of coverage for source code as a literary work. The essential question was whether and how traditional copyright doctrines for literary works applied to parse the protectable from the unprotectable elements of the source code. These doctrines included (1) the idea/expression dichotomy, under which copyright protects only the expression, not the idea; (2) not extending coverage to public domain elements; and (3) *scenes à faire*, a doctrine stating that protection does not extend to stock characters and common contexts. Eventually, courts worked out realistic formulations of these doctrines for source code as a literary work. What resulted is the well-known "abstraction-filtration-comparison" three-step test. Menell, *supra* note 2, at 82–85.

Through the first two steps, the test filters out certain elements from the source code, rendering these elements nonprotectable. What remains is then compared to an allegedly infringing work in the last step. Eliminated are code elements in the public domain, ideas, and items resulting from functional concerns, such as efficiency, specific code statements required under the operating system for which the program is written, and similar items. *Id.* The test recognizes that software, and source code, is more complex than the sequential series of statements in a typical recipe. It implicitly recognizes that what instead exists is an instructional composite, with its result-producing sequence and hierarchy of commands, routines, data, and interfaces.

[70]*See* 17 U.S.C. § 117(a) (2000) (authorizing owners of copies of computer programs to make certain types of additional copies for limited purposes).

[71]MAI Sys. Corp. v. Peak Computer, Inc., 991 F.2d 511, 519 (9th Cir. 1993); *see also* Joseph P. Liu, *Owning Digital Copies: Copyright Law and the Incidents of Copy Ownership*, 42 WM. & MARY L. REV. 1245, 1258–63 (2001) (discussing implications of RAM copies doctrine).

Thus, copyright offers a panoply of rights relevant to the open-source approach. The first is literal copying of the source code instructional composite. The second is creating RAM copies of the instructional composite, in both the source code and object code forms. Open-source users technically violate both of these rights merely by obtaining and running a copy of open-source software, at least for the expressive, copyright protected elements of the software. The violation is only technical because the open-source license grants a permission immunizing the violation as long as one follows the license conditions. The third is nonliteral copying of the source code, that is, copying the structure, sequence, and organization of the source code. The fourth is creating derivative works from the source code. Open-source developers who obtain open-source software and modify it may technically violate the third and fourth right. The fact of a technical violation in these rights is much less certain than the first two rights. This is because the doctrines applicable to nonliteral copying or derivative works violations are much less susceptible to generating certain outcomes when courts review these issues. In contrast, literal copying, when well proven, makes for a more predictable outcome.

All these rights, and potentially other rights protectable by copyright, are, paradoxically, the foundation of the innovative open-source approach. Copyright rights attach upon fixation. That is, from the moment the code is originally authored, to the extent it contains copyrightable subject matter, i.e., original expression, the protection attaches.[72] The open-source approach leverages this protection to extend "additional" control over the work. But the "additional" control imposes conditions that effectively ensure that the work is freely usable. This is the antithesis of traditional copyright-based licensing controls. But it is the innovation upon which the open-source approach is based. It uses the control of copyright to ensure that the collaborative aspects of the software persist: source code availability and royalty-free use.

Although copyright is the dominant traditional intellectual property regime applicable to software, it is not the most recent entrant to the scene. That distinction belongs to patent law, where the protective rights do not attach automatically as in copyright, but, when they do apply, are more powerful in certain, particularly important ways.

3. Patent Protection

Patent protection for software occurs in two general ways. These spring from the statutory definition of patentable subject matter: "Whoever invents or discovers any new and useful [(1)] process, [or, (2)] machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain

[72]17 U.S.C. § 102(a) (2000). Of course, registration of the copyright is advisable, and necessary in certain enforcement situations, but the burden of copyright registration is slight compared to the efforts required to keep a trade secret or apply for patent protection.

a patent therefore, subject to the conditions and requirements of this title."[73] Patentable subject matter is the first of five patentability requirements, the other four of which I will not discuss except in passing. Once software was judicially determined to qualify as patentable subject matter, like any technology, any particular computer program still needs to meet the other four requirements for a patent to issue.[74] Following the statutory definition, patentable subject matter is thought to fall into two broad categories: process patents and product patents.

The product patent category encompasses the three statutory terms "machine, manufacture, or composition of matter." Under this category patent rights would attach to software when the software was part of a product. This most often occurs when the software controls a machine or implements aspects of an apparatus.[75] For example, the office copier that you may regularly use could very well contain software covered by patent protection.[76] The office copier software likely contains only the object code instructional composite for the software. Even so, a product patent can cover such software within the context of the machine or apparatus. These patents cover the particular combination of the machine or apparatus, in cooperation with functionality expressed by the software. This functionality must be disclosed in the patent document, but is typically not given in source code. Rather, most patents employ a stylized or symbolized expression of the software's "routines" or algorithms.[77] In other words, the patent document discloses a stylized expression of the high-level instructional composite, used with the machine. Harkening back to the Bavarian-style coriander chicken recipe analogy, the stylized expression would be like a summary or diagram illustrating the recipe. Thus, the stylized algorithmic expression is often "higher-level," meaning more abstract, than the programming language itself. Until the late 1990s, product patent coverage was the primary mode of software patent protection available.

---

[73]35 U.S.C. § 101 (2000).

[74]Along with meeting the patentable subject matter requirement, an invention needs to meet the following four additional requirements to obtain patent protection: (1) utility, (2) novelty, (3) non-obviousness, and (4) disclosure requirements. 35 U.S.C §§ 101–103, 112 (2000).

[75]GREGORY A. STOBBS, SOFTWARE PATENTS xxix (2d ed. 2000) ("Software has broken free of its containment vessels and has leaked into everything: your VCR, the transmission of your car, the typesetting equipment that printed this book, and even the air traffic control system guiding your next landing.").

[76]For example, see U.S. Pat. No. 6,137,640 (issued Oct. 24, 2000), claiming a beam focus adjustment apparatus for image-setting equipment, in which the claim includes elements for a computer program to adjust the focus of a light beam based on the sensed temperature of the copying medium.

[77]STOBBS, *supra* note 75, at 277–78 ("In patent application practice, source code provides too much information and may not adequately identify what is new and inventive from what is old and commonplace.").

Under the process patent category, since the late 1990s, a computer program can qualify outright as patentable subject matter. The machine or apparatus context is not necessary. Thus, the patent covers an algorithm, as long as it is specific enough to produce a concrete, tangible result, and does not fall into one of a few very narrow exceptions. Process patents written to cover software freed from the machine or apparatus context are potentially much broader than product patents incorporating algorithm coverage.

With broader coverage,[78] process patents are troubling for the open-source approach.[79] As with patented technology generally, software that infringes a valid process patent does so even if the software was independently created. This circumvents the open-source copyright-based license. Users of the software comply with the license by observing the conditions. But an unrelated third-party can hold process patent rights that block open-source users from running the software.[80] To run the software would mean infringing the patent. Since the mid-1990s, software patent applications in the U.S. Patent and Trademark Office have grown dramatically.[81] Thus, open-source-blocking patents could exist now. In addition, third-parties could strategically obtain patents to competitively disadvantage open-source products.[82]

Process patents raise these issues for all software generally, but the proof of infringement is more easily obtained with open-source software because the source code is available. The very feature that makes the open-source approach attractive and beneficial puts it front and center in the target for patent infringement.[83] The object code instructional composite, when executing in the computer, is the action that infringes the process patent. But traditional software rarely distributes the source code with the object code. As a result,

---

[78]For example, assume that a valid product patent covers software in a copier that efficiently displays on the copier's interface a digitized number, which counts up as the copier makes each copy. If the same software were used to display a counting-up number on a display showing the number of visitors to a Web site, the copier patent would not be infringed. Now, changing the example, assume that the software is covered by a valid process patent that generally claims protection for a specific, efficient algorithm to display a counting-up number. This is a broader patent because both the copier and Web site display may infringe the patent. Indeed, depending on how the process patent is drafted, a wide variety of display technology employing the software may infringe.

[79]Robert W. Gomulkiewicz, *After 30 Years, Debate Over Software Is Still Noisy: Do Current Laws Protect Too Little or Too Much?*, 25 NAT'L L.J., May 12, 2003, at S10 ("Programmers, particularly those from the open-source movement, expressed concern that software patents stifle, rather than stimulate, innovation.").

[80]*See* Bessen, *supra* note 5, at 26, 28–29.

[81]*Id.* at 27.

[82]*Id.* at 28–29.

[83]McJohn, *supra* note 5, at 49. McJohn argues, however, that systemically and over the long run, open-source code may inhibit patent infringement suits because open and available source code increases the chance that litigants will discover patent-invalidating prior art. *Id.* at 50–52.

without actually filing suit and obtaining discovery, it can be substantially more difficult to determine if traditional software potentially infringes.

Software patent coverage troubles the open-source approach in other ways beyond my source code focus in this Article. There are also other aspects of software patent protection that do not bear on computer program source code, and as a result are beyond the current discussion. The critical point is that patent protection circumvents the copyright-based license. Patent protection allows circumvention via blocking rights under a different mode of traditional intellectual property protection.[84]

Recent history has witnessed an upswing in software patents, so the risk of open-source shackling is real, growing, but latent. As the open-source approach picks up more software projects and products, the potential grows for collision between the systems. As the human-readable instructional composite, source code orchestrates the computer program that delivers the desired computing result via the operating system and hardware. There is tremendous flexibility to implement algorithms in source code in a variety of ways. But that flexibility is not unlimited. The range of flexibility decreases as the number of software patents in an algorithmic area increases.[85] Thus, besides the threat specific patents may pose to specific open-source projects, software patenting may generally impede growth of the open-source approach by taking up "algorithm space."

---

[84]How does patent protection, which circumvents the openness of the open-source approach, work for software? The source code implements an algorithm or "recipe" which may be protected by patent rights. Software patents are written to cover specific algorithms, in specific contexts, that deliver certain tangible results. Even with this specificity and context, they can obtain very broad coverage. The patent instrument ends with a series of "claims." Patent claims are the legal definition of the holder's right to exclude. Typically the claims describe the invention in a variety of ways, sometimes claiming the invention broadly, sometimes narrowly. A process claim will describe the algorithm in English-language statements. The claim may employ terms of art for skilled programmers, but the claim will not be a direct recitation of the computer program's source code. Indeed, this would be a poor approach by the patent attorney because it would result in more narrow coverage. A general, technological description of the context-specific, tangible-results-producing algorithm provides much broader coverage. Such a patent claim could cover a software instructional composite implementing the algorithm written in either the "C" or "C++" or "Pascal" language. Indeed, it is not strictly necessary for the patent applicant to supply any source code with the application disclosure, so long as the disclosure provided enables a person skilled in the relevant programming art to practice the invention. In other words, the patent must describe (legally speaking, "claim") the "process" that it protects. When this process (the claim) maps to a software instructional composite, both its source and object code are effectively shackled if the patentee enforces the patent.

[85]*See* Bessen, *supra* note 5, at 13 (describing problem of "patent thickets" and their potential to inhibit open-source software).

Among the four major types of intellectual property, patent protection is most at odds with the open-source approach. Trade secret protection is inherently inconsistent with open-source software. Trademark law is an important adjunct, performing its traditional function to protect source-indicating significance for particular open-source software products, or to support certification programs that software meets some accepted definition of "open." In contrast to the other three areas, copyright law provides the foundation for open-source software. It implements, through generally applicable conditional licenses, the central tenets of the open-source approach, including source code availability. This enables collaboration, which occurs more effectively when programmers share the source code and implement computer program functionality with the understanding that all others can see and review the source code.

III. OPEN-SOURCE SOFTWARE

The growth of open-source software has been impressive. Supported by popular use in key Internet-infrastructure applications, the movement has achieved technological, social, and commercial successes. Grounded in the foregoing discussion of computing's three elements and the traditional intellectual property regimes bearing on source code, this Part highlights the history and nature of the open-source approach. This Part further sets the stage for the sections that follow, examining the central conditions that define the open-source approach, and then expressing a form of Collaborative Integrity for software, reminiscent of authors' and artists' right of integrity in civil law systems.

A. The Open Source Approach

Ideology sparked the open-source movement.[86] Programmers found benefit in the open-source approach and flocked to it, both for those benefits and in kinship with its underlying ideology. This ideology flipped traditional copyright. Instead of using copyright to control, the open-source approach used it to require freedoms. This movement toward open-source freedom corresponded with the rise of the Internet and its various strains and claims of cyber-freedom.[87] As programming talent collected around the open-source approach, projects appeared. Then, some projects grew into products, some widely used. Paradoxically, this "free" software movement then attracted

---

[86]*See* Stallman, *supra* note 45, at 55–56.

[87]*See* LESSIG, *supra* note 44, at 24–25 (introducing an argument to debunk common notion that cyber-space and the Internet are inherently and permanently "free" or not able to be regulated).

investment, entrepreneurs, and support from many well-known information technology companies.[88]

The open-source approach also exemplifies private provision of a public good.[89] The approach may generally define a "recipe" for private provision of certain public goods. Certainly, a wide variety of important software technologies are available as public goods due to the open-source movement. Some commentators have abstracted the open-source approach into a more general model of public-good provision.[90] This aspect of the open-source movement is an important feature of its mainstreaming, both commercial and otherwise.[91] But this mainstreaming would not have occurred without the originating ideology that sparked the movement.

---

[88]*See* Brian Behlendorf, *Open Source as a Business Strategy*, *in* OPENSOURCES, *supra* note 2, at 149, 149–52 (arguing that open-source software is "indeed a reliable model for conducting software development for commercial purposes" when the open-source project is a technology platform, because the open-source platform provides greater market growth due to standardization); 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 B6 (noting that "major 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"); Josh Lerner & Jean Tirole, *The Simple Economics of Open Source* 35, *at* http://papers.nber.org/papers/w7600.pdf (last visited Feb. 12, 2004) [hereinafter Lerner & Tirole, *Simple Economics*] (describing possibility for companies to benefit indirectly from open-source project by operating in complementary proprietary market segment).

[89]Lerner & Tirole, *Simple Economics*, *supra* note 88, at 2 (questioning why "thousands of top-notch programmers contribute freely to the provision of a public good"); Lawrence Lessig, *Open Source Baselines: Compared to What?*, *in* GOVERNMENT POLICY TOWARD OPEN SOURCE SOFTWARE 50, 56–59 (Robert W. Hahn ed., 2002), *available at* http://aei-brookings.org/admin/pdffiles/phpJ6.pdf (last visited Feb. 12, 2004) (discussing public goods theory in context of software); Siobban O'Mahony, *Guarding the Commons: How Community Managed Software Projects Protect their Work* (2003), at 3, *at* http://opensource.mit.edu/papers/rp-omahony.pdf ("Open-source software shares some similarities with privately produced pure public goods, but also differs from traditional definitions of public goods in important ways."); David P. Myatt & Chris Wallace, *Equilibrium Selection and Public-Good Provision: The Development of Open Source Software*, 18 OXFORD REV. OF ECON. POL'Y 446, 448 (2002) ("[O]pen-source software is a classic example of a pure public good.").

[90]*See* Benkler, *supra* note 4, at 369, 383, 434–36 (postulating peer-production model as alternative to organizing production according to markets or firms). In a larger discussion of the trade-offs for weak versus strong intellectual property rights, in which open-source software is given as an example of a beneficial weak-intellectual-property-rights regime, one commentator notes that "[l]egal practice itself might offer a nice illustration of the promise of progress in spite of weak property rights. New legal arguments are cited, copied, and exploited as soon as the imitator likes, and yet there is no apparent shortage of brief writers or of talented persons entering the field of law." Saul Levmore, *Property's Uneasy Path and Expanding Future*, 70 U. CHI. L. REV. 181, 185 (2002) (citation omitted).

[91]*See* Behlendorf, *supra* note 88, at 150–52 (noting importance of freely available and standardized technology platforms to create commercial opportunities around open-source software).

1. Ideological Origins

The open-source approach values source code. It finds software more valuable with source code. It expresses these values in response to the command-and-control programming methodologies that early computing developed in response to a variety of factors. It codifies these values in a copyright-based licensing approach. A few widely used licenses exemplify the approach. But it is potentially extensible to support a variety of values. Ultimately, the open-source approach rests on the licensing power to impose conditions on the use of a work. If a user meets the conditions, she has permission. This Subsection reviews the origins of open-source values, and sketches their implementation.

(a) Early Computing and Software Development

Given their incentives, companies tend to keep secrets, including computing secrets, at least until disclosure has a benefit or potential benefit. Rebellion against this practice is an originating factor for the open-source movement.[92] Source code secret-keeping, and its anti-collaborative effects, however, became endemic to early computing due to two factors present from the start of computing through the early 1980s: the centralized nature of the technology[93] and its increasing commercialization. The centralized computing design was a technological constraint during this era. This constraint, however, contributed to a regime of control over the computing assets, especially as their commercial value grew.

Corporate law requires companies to shepherd assets and employ them productively. As computing assets grew in importance, so did company practices to restrict access to various aspects of the technology, including

---

[92]Harry Rubin & Jason Isaacs, *The Myths and Realities of Open Source Code Licensing: Business and Legal Considerations*, 8 No. 3 CYBERSPACE L. 2 (2003). The authors describe the impact of open-source development as follows:

> [O]pen source development is a challenge of taboo-breaking dimensions. Software development has traditionally been conducted in acute secrecy and subject to strict confidentiality, development, proprietary and non-disclosure agreements. Source code, always considered the "crown jewel," has been vigorously protected. Most source code disclosures occur only after a narrowly defined source code escrow release condition has been triggered. Open-source licensing now requires software companies to eschew the very axioms which have governed software since the dawn of the industry and adopt the polar opposite approach.

Id.

[93]The computers of these past eras required special facilities and care. Accordingly, companies housed their computing equipment in secure and central locations. They granted access to programmers, users, and administrators through "dumb" terminal devices with no computing power or graphical capability by today's standards. Users and programmers interacted with the remotely located computer via the terminal and perhaps a printer.

physical access to the machine, as well as user access to the operating system and the source code of computer programs on the machine. Reducing access decreased operational disruption risks. Information technology and the computing assets became more valuable and mission critical. As a result, companies ratcheted up the control over these assets. Operating systems responded to this need by offering a hierarchy of user levels, allowing administrators into the system with full "power" to change and configure the system, while restricting users and programmers to specific environments within the hierarchy. This was the era when "big iron" dominated computing and defined the era of centralized computing. The processor was secreted away somewhere, access was via terminals, and only a few administrators had the power to range throughout the entire system. This hierarchy of control, it was thought, better protected the corporate assets. It also facilitated secret-keeping for source code.

Smaller computers (predominantly personal computers) and communications technology (primarily computer networks) broke the "big iron" centralized computing mold and ushered in the era of distributed computing. Networking technology led to the Internet, an interconnection among networks (or computers) based on standard protocols. Now every programmer had her own processor, her own computer.[94] As networking grew, she was connected with an increasingly large community of technologists. This technological change of the 1980s through the present provided fertile soil for the early proponents of the open-source approach.

That approach was characterized by a belief in the inherent value of making source code available, of keeping the source code free.[95] Source code is a text to be shared, read, and studied. It should be treated as the literature of computing technology. Like literature, its dissemination enlightens minds and teaches lessons of success and failure. It facilitates collaboration, augmenting, and leveraging the works of others. Freely available source code made programming more fun and made software transparent, and thus more valuable.

This approach was rebellion because, like the "big iron" under their control, companies also needed to control the groups programming the "big iron." These groups were human capital, assets from which companies also needed to wring profits given the corporate mandate. The management practices that developed to fulfill this mandate embodied traditional corporate command and control over the software development process.[96] A hierarchy provided the structure for command and control. Spots in the hierarchy

---

[94]BROOKS, *supra* note 3, at 281–82.

[95]*See* Eben Moglen, *Anarchism Triumphant: Free Software and the Death of Copyright*, 4 FIRST MONDAY, Aug. 1999, *at* http://www.firstmonday.dk/issues/issue4_8/moglen/index.html ("[T]urning software into property produces bad software . . . .").

[96]*See* BROOKS, *supra* note 3, at 35–37 (describing programming teams).

determined specialization of function. The user hierarchy in the operating system helped enforce the organizational hierarchy among the programmers. It also could partition the source code on large projects. The user hierarchy would give the programmer access only to specified sections of project source code.[97] To the open-source progenitors, this corporate command-and-control approach was, at best, counterproductive. It devalued source code. The teachings of the code were unavailable. At worst, the progenitors found the hierarchical software development environment creativity-stifling and inapposite to professional freedom.

"Big iron" computing and corporate organizational techniques caused the traditional, hierarchical software development process, but in its course, the open-source approach fermented in opposition to the allegedly ingenuity-destroying aspects of the source-hiding hierarchical regime. This regime hid some of the source from some of the programmers in the typical large software project. But it hid all of the source from all of the users. Customers and end users, whether they accessed the software remotely, or ran it on their own computers, typically did not have access to the source code. This, from the perspective of those in the open-source movement, was also counterproductive. The ferment against the traditional hierarchical regime resulted in a copyright-based licensing approach with a goal to require source code availability. This licensing approach expresses and codifies open-source values and benefits. Later in this Article, I explore the licensing conditions in more depth. In order to facilitate the remaining discussion of open-source's history, nature, and status in this Part, however, the next Subsection sketches the approach in more detail than heretofore discussed.

(b) Sketching the Open-Source Approach

The open-source approach employs licensing to achieve its aims. Many different licenses are used for open-source software. One license has particular prominence due to its author and popularity.[98] Richard Stallman is a progenitor of the open-source movement and author of the General Public License ("GPL")[99] This Subsection will discuss the GPL as a paradigmatic example of

---

[97]*See* Sanne te Meerman, *Puzzling with a Top-Down Blueprint and a Bottom-Up Network*, *at* http://opensource.mit.edu/papers/temeerman.pdf (Feb. 2003), at 16, 17 (describing management, organization and coordination differences between Linux development team and Microsoft Windows development team).

[98]Josh Lerner & Jean Tirole, *The Scope of Open Source Licensing*, *at* http://papers.nber.org/papers/w9363.pdf (Dec. 2002), at 23 [hereinafter Lerner & Tirole, *Scope of Licensing*] (concluding from survey of popular Sourceforge.net open-source project repository, holding almost forty thousand projects, "the dominant role of the General Public License" was apparent because approximately three-quarters of projects used the GPL).

[99]*See* GNU, GNU General Public License, *at* http://www.gnu.org/licenses/gpl.html (last visited Feb. 12, 2004) [hereinafter GNU, GPL] (displaying General Public License).

the open-source approach. Other licenses, however, protect open-source software with varying differences.[100] The scheme uses copyright. Without the conditional permission in the open-source license, the user faces copyright infringement.[101]

Thus, the open-source approach rests on the licensing power to impose conditions on the use of a work. If a user meets the conditions, she has permission. Using the GPL as an example, what use rights are allowed, and what are the conditions defining what I call the open-source approach? In broad terms, the license allows one who takes the software to use it, modify it and redistribute it if she (1) makes the source code available, (2) does not charge royalties for software use, (3) propagates the same terms for redistributed or modified software, (4) includes notice of the GPL terms, (5) attributes modifications to the maker, and (6) disclaims warranties and liabilities.[102]

To illustrate the GPL's effect on software licensed under it, consider the following example. Assume that Allen creates a program we will call "GoneFishing$_A$" that searches the Internet specifically looking for information about fishing, allowing the user to specify the type of fishing. Allen publishes GoneFishing$_A$ on his Web site, and tags the program with notice that it is made

---

[100]GNU, *Various Licenses and Comments about Them*, *at* http://www.gnu.org/licenses/license-list.html (last visited Feb. 12, 2004) (listing dozens of licenses that implement to some degree open-source approach, and commenting as to whether licenses are compatible with GPL and/or meet Free Software Foundation's definition of "free" software); Open Source Initiative, *The Approved Licenses*, *at* http://opensource.org/licenses/index.html (last visited Feb. 12, 2004) (listing licenses under which distribution of software qualifies for use of OSI certification mark because OSI has determined that listed licenses meet its definition of open-source software).

[101]My focus in this Article is the open-source license as a document granting permission, rather than as a contract, although in a later section I review some of the issues raised by the open-source license as a contract. *See infra* section III.B.5. Practically, however, this distinction can be important, for open-source licensing and for licensing generally, in particular for the scope of remedies. *See* Sun Microsystems, Inc. v. Microsoft Corp., 188 F.3d 1115, 1122 (9th Cir. 1999) (holding that whether Sun could enjoin Microsoft based on license agreement from Sun to Microsoft for Sun's Java technology depended on whether license provisions in question were "license restrictions or separate [contractual] covenants"). My thanks to David McGowan for raising this point.

[102]GNU, GPL, *supra* note 99. The GPL has other terms, but these are the primary operational terms and are sufficient for the present discussion. The list has latent repetition for emphasis because the GPL itself disclaims warranties and liabilities. Thus, propagating the same GPL terms for redistributed or modified software disclaims warranties and liabilities for such software. To ensure that sufficient source code is available with the software so that others can compile it and produce the object code executable instructional composite, the GPL provides a specific technical definition of the source code that must be made available. This definition recognizes that the computer program is a composite of instructions, data, and interfaces in a sequence and hierarchy. This instructional composite will produce a particular desired computing outcome, directly as object code running in the computer, but which is indirectly scripted by the source code.

available under the GPL. If Betty downloads a copy, she can use it under the GPL terms. Betty determines that GoneFishing_A is slow when searching for information about deep-sea sport fishing. Having the source code from the download, she adds new routines for faster deep-sea sport fishing searches. But Betty also likes deep-sea kayaking, so she programs her new routines to optionally search for kayaking information. Betty has created a new source code instructional composite, with some of Allen's GPL source code, and some of her new source code. To take this step, merely creating the modification, in compliance with the license, Betty needs to attribute to herself the new routines she added to GoneFishing_A. This is done through the comments that programmers can embed in the source code instructional composite. If she merely uses for herself her new version of the program, "GoneFishing_B"; she need do no more to comply with the license.

Next, assume that Betty posts GoneFishing_B to her own Web site, inviting others to download it. Under the GPL, if she charged royalties for use of GoneFishing_B, she would contravene the license terms. To comply with the license, she must make all of the source code for GoneFishing_B available, including both hers and Allen's. She must ensure that the software includes notice that the GPL terms apply, which propagates the same terms to takers of GoneFishing_B, including the GPL's warranties and liabilities disclaimer.

Finally, assume that Carol downloads GoneFishing_B. Carol reprograms part of it and adds new routines. Carol calls her program "GoneOutdoors" because it searches for information about any outdoor sporting activity. Her new code is intermingled with Allen and Betty's code. Despite the intermingling, if Carol attributes, through commenting, her changes and additions, and complies with the other GPL terms, she may distribute GoneOutdoors as open-source software. The mere fact of having changed the name for the software does not change the GPL applicability. Carol makes GoneOutdoors available on her Web site. Various users download and run it, including Allen and Betty. The example does not need to stop at three iterations of the software, from GoneFishing_A to GoneOutdoors, but it shall, the point hopefully being made: the open-source approach allows the sequence to continue along a chain of developers, or, within a web of "cross-using" developers. Viewed through the three elements of computing developed above, the figure below conceptualizes the GoneOutdoors software.

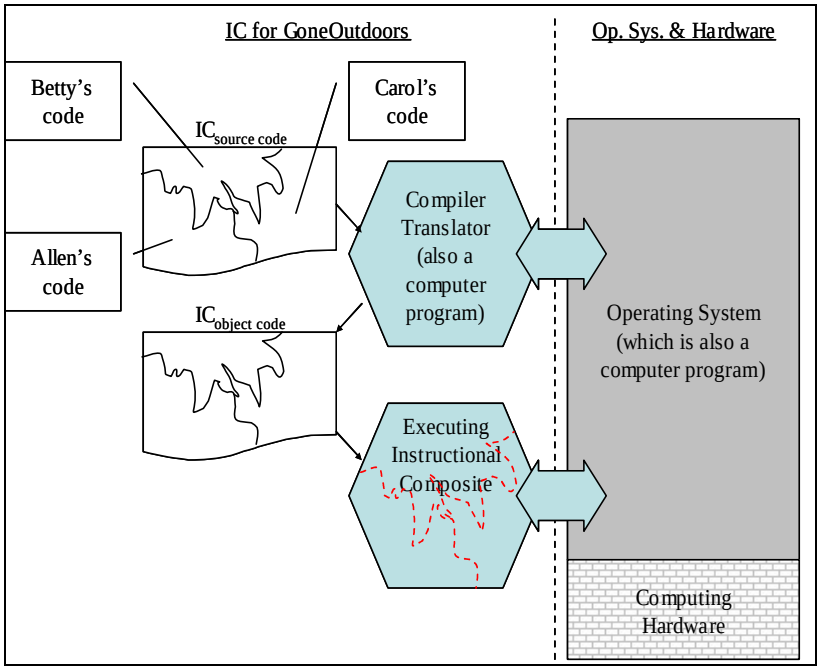

Figure 2

Contributed Code to the Hypothetical "GoneOutdoors" Open-Source Software

In this example, three user-programmers successively constructed GoneOutdoors. This raises a variety of intriguing copyright questions, including issues related to authorship, derivative works, and the potential scope of source code nonliteral infringement.[103] Another set of issues is the role of contract doctrine, if any, in the chain (or web) of GPL permissions or commitments.[104] Licensing law is sometimes characterized as a species of contract law. With the open-source approach, however, the license involved

[103]For authorship, there is a "complex, often unclear body of law dealing with joint ownership," which is particularly difficult to determine rights when two or more persons or organizations collaborate to develop software. NIMMER, COMPUTER TECHNOLOGY, *supra* note 62, § 4:15, at 4–32. Without the permission in the license, the downstream modifiers of the open-source project might infringe the original holder(s) derivative work right along with the reproduction right. Traditional analysis of the derivative work issue may be strained because "digital technology changes the rules and scope of the issue." *Id.* §1:104, at 1–274. Finally, besides direct literal infringement by downstream users, because users undoubtedly employ some of the original code unadulterated, their modifications, but for the license, might also require application of legal tests for nonliteral infringement of source code.

[104]*See* McGowan, *supra* note 4, at 289–302 (discussing legal status of GPL license from perspective of contract law and identifying variety of issues that may arise).

may spring from other areas rather than strict contract doctrine. These issues are explored later.

Several features of the open-source approach are more important to take from this initial sketch. First, one who takes and uses the software must fulfill enumerated obligations to satisfy the license's conditional permission to use. Second, modifying and redistributing the software triggers a more expansive set of conditions. Third, the conditions emphasize and enable successive, collaborative development of the software. As with GoneOutdoors, multiple programmers can contribute to the source code. And, they can each become users of the other's modifications, generating a copyright ownership and licensing web.[105] Fourth, the successively built web of ownership is a unique by-product of the open-source approach. These features highlight the innovative and novel characteristics of the approach. They are also, in addition to the values the approach embodies and expresses, a substantial determinant of the growth and success of various open-source products.[106]

Open-source collaboration, expressed by the GPL as a freedom to share and change "free" software, implemented under the open-source approach, has proven thus far to tap and release substantial programming energies. This release, often given as volunteer effort, has produced important software components underlying the Internet. The next Subsection will review the paradigm-challenging effect these open-source software products have had on traditional software development.

---

[105]McGowan, *supra* note 4, at 259.

[106]These values are expressed in the preamble text of the GPL:

> The licenses for most software are designed to take away your freedom to share and change it. By contrast, the . . . General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users . . . .
>
> When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

GNU, GPL, *supra* note 99.

2. Projects and Products

(a) The Growing Trove of Open-Source Software

Open-source software ranges across all types of computer programming projects. There is a great variety and quantity of projects.[107] One could classify the vast variety of open-source projects along many metrics. One obvious metric would be the number of developers.[108] Another would be the scope of the software project or product. Is it a large all-encompassing program of the sort that often have millions of lines of code, such as an operating system or a suite of personal productivity applications, or a relatively small component of a larger software product, such as a report generator for a programming environment? A third metric would be its technological aspects: what operating system is it for (assuming it is not an operating system itself), what type of application is it, or what source language is it written in? Another classifying approach would be the targeted users of the software.[109] Some software envisions technologists as its end users. Other software, by nature of its intended function, envisions non-technologists as the primary end users.[110]

---

[107]*See generally* SourceForge, *What is SourceForge.net?*, *at* http://www.sourceforge.net (last visited Feb. 12, 2004) [hereinafter SourceForge, *What is SourceForge*] (describing this Web site's services to open-source developers in which Web site provides free "centralized place for Open Source developers to control and manage Open Source software development"). SourceForge hosts "tens of thousands" of open-source projects. *Id.* By "hosting" an open-source project, the SourceForge Web site provides a number of usefully aggregated capabilities. These include: (1) allowing users to download versions of and revisions to the open-source software; (2) customer support communications and tracking ability; (3) programming team communication and collaboration tools such as discussion lists and projects diaries; and (4) other tools related to compiling, promoting, enhancing, and managing the software over time. *See* SourceForge, *Services to Projects We Host*, *at* http://www.sourgeforge.net (last visited Feb. 12, 2004). The title of this Subsection, in particular use of the word "trove" is stylized from the "Trove, a massive database of Open Source projects." *Id.* The Trove is available for searching at: http://sourceforge.net/softwaremap/trove_list.php.

[108]Kieran Healy & Alan Schussman, *The Ecology of Open-Source Software Development*, *at* http://opensource.mit.edu/papers/healyschussman.pdf (Jan. 29, 2003), at 2, 9–13 (describing large scale survey of the SourceForge.net database of open-source software projects, and finding that most projects have small number of developers).

[109]*See* Lerner & Tirole, *Scope of Licensing*, *supra* note 98, at 21, 38 tbl.2 (classifying open-source projects along various metrics to determine whether certain types of projects were more likely to select certain types of licenses).

[110]The point about the available variety of open-source software, and the many ways to classify it, is two-fold. First, the variety demonstrates the popularity and success of the open-source approach in attracting programming talent to start and contribute to projects that, presumably, users or other programmers find useful, and for which, presumably, the participating programmers find interesting. Second, the variety of projects and products implies robustness in the open-source approach. Something about the approach must work well to release the latent energies that it has.

Two of the surmised metrics are related: small projects will have a small participating group,[111] but the opposite will typically be true for large projects. An example is GoneOutdoors. If Allen and Betty join Carol to enhance GoneOutdoors, collecting and issuing their code on and from Carol's Web site, with Carol acting as the group facilitator, their collaborative activity is an example of a small-group open-source project. If the product is valuable and more effective than competing software, a user base may emerge.

Open source, however, does not stop at small projects.[112] Indeed, it is large projects, most notably an operating system, Linux, and a Web-page server, Apache, that have catapulted open-source software to success and repute. With this success comes challenges, debate, and, to some, notoriety.

On the other hand, my point is only valid if taken in its broadest sense: the aggregate activity signals something useful, not that every open-source project is successful. The caveat is because empirical data show that most of the projects registered at the Sourceforge.net open-source repository are sole-programmer efforts with no additional contributors. *See* Kieran Healy & Alan Schussman, *The Ecology of Open-Source Software Development*, *at* http://opensource.mit.edu/papers/healyschussman.pdf (last visited Feb. 13, 2004), at 12 ("[O]nly a tiny number of projects have more than a handful of developers. . . . [T]here is little or no programming activity taking place on more than half of the projects."). There may be several ways to interpret the data, but it is clear that most open-source projects do not enjoy the success of projects such as Linux. The aggregate level of activity signals value in the approach, unless all contributors, even those authoring sole-programmer projects, are foregoing better opportunities for their time and energy. *See* Lerner & Tirole, *Simple Economics*, *supra* note 88, at 20–25 (discussing programmer incentives from economic perspective).

[111]Some projects are the hobbyist's sole output. The products from these projects may only ever be used by a few others. Sometimes, however, these products develop a popular following. Sometimes projects in the "hobbyist" category are learning labs for the programmer, a place to develop new skills and perhaps develop a user base for her software. Often the sole hobbyist programs open-source tools, utilities, add-ins, and other components that bring incremental value to other, larger software. In other cases, small groups gather virtually to develop such utilities and components.

[112]Indeed, it may be that the large open-source projects are the truly new phenomena, while the large number of small or sole-programmer projects are simply a variant of the "shareware" tradition that has long existed in the personal computer world. *See* Healy & Schussman, *supra* note 110, at 12 (noting that most open-source projects have one developer, or very small number of developers). "Shareware" is software that someone writes and offers to others, but only in executable, object code form, and typically with a request that a small license fee be paid. Matthew A. Liao-Troth & Terri L. Griffith, *Software, Shareware and Freeware: Multiplex Commitment to an Electronic Social Exchange System*, 23 J. ORG. BEHAV. 635, 638 (2002). Typically, however, neither the distribution method nor the software itself enforces this license. Users who take a copy of the software are on their honor to send in a payment. Alternatively, the shareware version has reduced functionality, and with a small licensing fee a user can obtain an "official" copy and full functionality. SCOTT, I&T LAW REF., *supra* note 50, at 712–13. Thus, one key difference between small open-source projects and shareware is that, traditionally, shareware was not provided with the source code. *See* Telephone Interviews by Rishab Aiyer Ghosh with Linus Torvalds (1996–98), *at* http://www.firstmonday.dk/issues/issue3_3/torvalds/ (last visited Feb. 13, 2004) [hereinafter Torvalds Interview] (discussing preference for open-source approach as opposed to shareware approach prevalent in personal computer world).

Each of these two products deserves special mention. One poses a credible threat to Microsoft's operating system dominance while the other serves up a majority of the Web pages viewed around the world on the Internet's World Wide Web.

(i) Linux: A Privately Provisioned Public-Good Operating System

Linux is an open-source operating system. Operating systems are important because they control the computing hardware.[113] They are a foundation technology for the rest of computing. The recent history of operating systems, in particular on personal computers, revolves around two stories. The first is the well-known story of Microsoft and its dominance in that market, particularly for "desktop" computers.[114] The second is becoming better known: the rise of Linux. Like Allen's GoneFishing computer program that evolved into a collaborative project among Allen, Betty, and Carol, Linux began as a single individual's code that attracted participants and grew under the open-source approach. The genesis was in 1991, when Linus Torvalds programmed an operating system "kernel."[115] Torvalds' original project was to explore a particular design for an operating-system kernel.[116] The kernel that would go on to become Linux was, in part, a project to comment and improve

[113]An operating system is itself a computer program. It is a special program that makes the computer hardware usable. Recall the cook in the kitchen analogy, where the Bavarian-style coriander chicken recipe was the software, the cook was the operating system, and the kitchen equipment was the computing hardware. The operating system models the computer hardware in software, and allows other programs to direct and command the hardware, but only through the operating system. In the cooking example, it is as if Francis, who owned the kitchen and employed the cook, was not allowed to go into the kitchen and do anything himself. He could only ask the cook to do it. This is what an operating system does to other programs that are commonly called applications. It makes applications ask the operating system ask the hardware to do work for them. The applications cannot access the hardware directly. They cannot go into the kitchen, they must ask the cook to do the work in the kitchen.

[114]*See* United States v. Microsoft Corp., 253 F.3d 34, 54–55 (D.C. Cir. 2001) (noting that under even broad operating-system market definition, Microsoft holds more than eighty percent of market, and that its Window's operating system "supports many more applications than any other operating system").

[115]BRYANT & O'HALLARON, *supra* note 46, at 18. Continuing the "cook is the operating system" analogy, in slightly squeamish fashion, the kernel of the operating system is perhaps like the cook's brain and nervous system. At the time he wrote the kernel and made it available to an online community of programmers, Torvalds was a graduate student in computer science at a University in Finland. *Id.*

[116]Linus Torvalds, *The Linux Edge*, *in* OPENSOURCES, *supra* note 2, at 101, 103–04.

upon certain design aspects of a family of operating systems generally known under the name "Unix."[117]

The traditions that developed in the Unix programming world set the stage for open-source development: sharing source code; collaborating on the design and development of software components and projects; effectuating the collaboration over the network; and loosely and informally directing, managing, and organizing the collaborative effort.[118] The Unix world traditions were amplified by the open-source approach. When this amplified force met Torvalds' kernel project, it propelled the project, under his leadership, and in combination with GNU project components from the Free Software Foundation, to a spectacular example of counterintuitive results on many fronts.[119]

---

[117]The Unix group of operating systems has been in existence and use since the early 1970s. They were primarily used on "mid-range" computers: machines more powerful than most personal computers, but less powerful than mainframe computers. Although very important to information technology, mid-range computers are much less numerous than personal computers. In today's parlance, sometimes the term "server" approximately corresponds to mid-range computing machines. They typically do not run Microsoft's Windows operating system, and thus, account for only a small share of the operating-system market across all of computing, measured by the number of machines. The history of programming traditions and activity in development of the various Unix operating systems presaged the open-source approach. One major "flavor" of Unix sourced from Berkley, which made the source code available, but under a license with less conditions than the open-source approach. There was also, to some degree, an "underground" of trading and taking source code from place to place. This resulted in a degree of informal, officially unauthorized, but perhaps tolerated sharing. Open Source Initiative, *OSI Position Paper on the SCO-vs.-IBM Complaint*, *at* http://www.opensource.org/sco-vs-ibm.html#id2791689 (last visited Jan. 27, 2004). Companies also undertook much official licensing of the dominant flavors of Unix. The end result of this evolution from the early 1970s to the late 1990s is that there were multiple popular flavors of Unix, all of them very similar, many of which shared a lineage to one of several common source code origins. The source code was available to many Unix technologists, both through informal means and via formal licensing.

[118]Marshall Kirk McKusick, *Twenty Years of Berkeley Unix From AT&T—Owned to Freely Redistributable*, *in* OPENSOURCES, *supra* note 2, at 31, 40 ("The history of the Unix system and the BSD system in particular had shown the power of making the source available to the users. Instead of passively using the system, they actively worked to fix bugs, improve performance and functionality, and even add completely new features.").

[119]As discussed in note 12, what is commonly called the Linux operating system is a distribution of software components. Besides the Linux kernel from Torvalds and the other kernel contributors, the distribution includes important components derived from the Free Software Foundation's GNU project. GNU is a "recursive" phrase because, as the name of an operating-system kernel, it is an acronym that stands for: GNU is [N]ot Unix. Early in his project, Torvalds decided to use the GPL for Linux in order to use the compiler available for the GNU project. Torvalds, *supra* note 116, at 107. The GNU project's goal was to build a free operating system, and it started before Torvalds wrote his initial kernel. The GNU kernel, however, progressed more slowly than Torvalds' Linux kernel. Stallman, *supra* note 45, at 64–66. As a result, the GNU/Linux combination, popularly called Linux, in effect took the first mover advantage as an open-source operating system.

Whatever technological innovation present in the kernel that became Linux was not the dominant reason for its growth and popularity. Rather, the reason was the open-source approach.[120] Being interested in feedback and input on his design, Torvalds shared the source code with others. Eventually, he decided to apply the open-source approach to his kernel, using the GPL. This led to interested participants sharing ideas and expressing an interest in further extending the kernel source code. Torvalds' kernel project took off and he continued to play a coordinating leadership role as literally thousands of programmers over the years developed and extended the kernel into a full-fledged operating system.[121]

Linux is a leading success story of the open-source movement.[122] Strikingly, through private efforts, Linux provides a resource that is, in essence, being offered as a public good.[123] Open-source software has the potential to privately provide public goods on a grand scale. Linux's impact already reaches that benchmark. Linux, like software in general when an abundance of computing power is available, exhibits non-rivalrous properties.[124] As a work covered by copyright protection, excludability is a possibility. But, under the open-source approach, excluding others' use has been mostly disclaimed. The conditional permissions underlying the open-source approach encourage others' use. Among other items, the conditions impose source code availability and prohibit royalties. The conditions do not negate non-excludability, but may make it less pure because, for behavior that contravenes the open-source license, there is excludability. But this is a small

---

[120]*See* Torvalds Interview, *supra* note 112 ("Making Linux freely available is the *single* best decision I've ever made. There are lots of good technical stuff I'm proud of too in the kernel, but they all pale by comparison.").

[121]Here I continue the "cook is the operating system" analogy. In essence, starting with the kernel, that is, the cook's brain and nervous system, the Linux group participants gave the cook the rest of his anatomy, a torso and appendages, senses and internal organs, a face, and a smile.

[122]In part, Linux's success is a function of branding. Torvalds obtained a U.S. trademark for "Linux" which serves to designate open-source operating-system distributions including his kernel. Many open-source projects use a trademark to identify the project. Ruben van Wendel de Joode et. al, *Protecting the Virtual Commons*, *at* http://opensource.mit.edu/papers/joode.pdf (Sept. 2002), at J5. The similarity of "Linux" with the term "Unix" is taken to expresses appreciation for the inspiration that Linux owes Unix. In addition, the GPL itself may exhibit a trademark-like effect, signifying that a project is open-source because it employs for protection the same, well-known license used by Linux.

[123]Myatt & Wallace, *supra* note 89, at 447–48; Eric von Hippel & Georg von Krogh, *Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science*, 14 ORG. SCI. 209, 209–23 (2003), *available at* http://opensource.mit.edu/papers/hippelkrogh.pdf (discussing collective action model for private provision of public good and its application to Linux).

[124]Also assuming, of course, that the software does not depend on some other constrained resource, such as a database that can serve well only a limited number of users, or a communication channel with limited bandwidth.

range of behavior compared to the dominant activity—using the open-source software.[125]

Other measures signal Linux's success. Estimates put Linux's share of the overall operating-system market, which by sheer numbers is dominated by "desktop" computers running Microsoft's Windows operating system, at two percent, but growing.[126] In the submarket for "server" computers, which underlie much of the Internet, the estimate is approximately sixteen percent.[127] Commercial mainstreaming, that is, entrepreneurs and companies seeking to profit by establishing new (or old) business models on the open-source movement, has primarily focused on Linux. It is an operating system, so it is foundation technology. As a result, many view it as a threat to Microsoft's dominance in the operating-system market and other areas of computing.[128] This alone creates commercial opportunities for Linux and other open-source software.

Linux's royalty-free "price" also has many large entities interested, including governments—as users—around the world. Although a variety of other factors contribute to a user's cost-of-ownership for any particular technology,[129] the royalty-free nature of Linux and much open-source software has many users planning to use it more, in order to reduce information technology costs.[130] Many government organizations have encouraged open-

---

[125]In the sense of direct use, the open-source approach has complete non-excludability. The license allows such use. While anyone can use, one contravenes the license conditions by charging someone else for use of the software in original or modified form. Thus, commercialization of the software is an excluded activity under licenses such as the GPL. There may be opportunity costs in using open-source software, such as foreclosing the opportunity to later charge royalties on software commingled with open-source code, but this is not tantamount to a prohibition excluding use. *See* Lerner & Tirole, *Scope of Licensing*, *supra* note 98, at 3 (discussing so-called "viral" characteristic of GPL). *See also* Eric S. Raymond, *The Magic Cauldron*, *at* http://catb.org/~esr/writings/magic-cauldron/ (June 1999), §§ 7–9, at 4–7 (discussing use value versus sale value of software, and noting that most software is developed for its in-house use value).

[126]Margret Johnston, *IDC: Microsoft Tightens Vise on OS Market*, *at* http://www.computerworld.com/governmenttopics/government/legalissues/story/0,10801,58278, 00. html (Feb. 28, 2003).

[127]Robert A. Guth, *Free to Choose*, WALL ST. J., May 19, 2003, at R6. In the server market, Linux shipments are estimated to climb to 15.9 percent in 2003 from 13.3 percent in 2002. Microsoft's estimated shipments, however, were essentially flat from 2002 to 2003, at approximately 60.4 percent. *Id.* Other estimates, however, put Linux's percent share of the server market in the mid-twenties. Johnston, *supra* note 126.

[128]Guth, *supra* note 127, at R6. ("Linux may be the biggest threat Microsoft faces. Even Microsoft, at last, seems to recognize that.").

[129]David S. Evans, *Politics and Programming: Government Preferences for Promoting Open Source Software*, *in* GOVERNMENT POLICY TOWARD OPEN SOURCE SOFTWARE 34, 42 (Robert W. Hahn ed., 2002), *available at* http://aei-brookings.org/admin/pdffiles/phpJ6.pdf (last visited January 8, 2004) [hereinafter Evans, *Preferring OSS*].

[130]Bulkeley, *supra* note 88, at R6.

source use. Some have even mandated its consideration in government procurement regulations.

Operating systems like Linux make the computer a computer—they are a critical foundation technology. Linux has emerged remarkably as a potential competitor to Microsoft in operating systems. Equally notable, it is a replacement technology for (while being inspired by) the flavors of Unix that have dominated mid-range computing for the past three decades. For these and many other reasons, it is the name bearer for the open-source approach.

I continue the review of key open-source products in the next Subsection with one additional example: the Apache Web server. Like Linux, it has a clear product market space, competes vigorously and successfully with Microsoft's offerings in this product space, and is a fundamental technology—in this case for the Internet's World Wide Web. The Apache project also demonstrates an open-source software license with minimal restrictions. If the GPL is one end of the continuum for open-source licenses, the Apache license is the other extreme.[131]

(ii) The Apache Web Server: Unrestricted Open-Source Software

Web server software implements the hypertext transfer protocol ("HTTP") and associated protocols to "serve up" to Web-browser users the multimedia pages that define the Web experience.[132] The Apache project began with source code from an early Web server developed by the National Center

[131]A variety of licenses exist along the continuum. *See* Steve H. Lee, *Open Source Software Licensing*, *at* http://cyber.law.harvard.edu/openlaw/gpl.pdf (Apr. 28, 1999), at 50–57 (summarizing terms of several licenses, describing "battles" within open-source community as to which license is best, and arguing that "the question of which open-source licensing scheme is better boils down to . . . finding the best means to achieve the ultimate goal of sustaining and advancing the open-source development model").

[132]Products like Apache make the Web the Web, just as operating systems make a computer a computer. Web server software sends information from a "server" computer to a user's browser that has requested the information, typically by clicking on a link. The server computer is special primarily in that it runs the Web server software and has the information underlying the Web page(s) available to it. Thus, the Web server computer could run one of Microsoft's Windows operating systems, and also run Microsoft's Web server software. Alternatively, the computer might run Linux and Apache.

for Supercomputer Applications at the University of Illinois ("NCSA").[133] A group of Webmasters in the mid-1990s, using the freely available NCSA source code, decided to develop the software to their liking.[134] They did so at first as a loosely organized group, but later formed an entity: the Apache Software Foundation ("ASF").[135] As a foundation, the ASF could implement formal structures for governance and management of projects.[136] Foundation leaders also determine when to accept and incorporate submitted changes back into the "official" ASF version of the software.[137] Apache users who submit changes back to ASF may do so for a variety of reasons, including to fulfill community norms or to ideologically support the open-source movement, but also for the practical consideration that it makes management of their installation easier. A user will typically want the other modifications made to Apache in future versions. If her modification is one of these, then she eliminates the otherwise required effort to back-fit her changes into each successive new version.[138] The Apache project has been very successful and, as

---

[133]Lerner & Tirole, *Simple Economics*, *supra* note 88, at 10; Mockus et al., *supra* note 4, at 316. Raymond provides a motivation for the Apache group's decision to collaborate and build their own Web server:

> The Apache server was built by an Internet-connected group of Webmasters who realized that it was smarter to pool their efforts into improving one code base than to run a large number of parallel development efforts. By doing this they were able to capture both most of the advantages of roll-your-own and the powerful debugging effect of massively-parallel peer review.

Raymond, *supra* note 125, § 7.1.

[134]*See* Lerner & Tirole, *Simple Economics*, *supra* note 88, at 10.

[135]Apache Software Foundation, *Frequently Asked Questions: Answers*, *at* http://www.apache.org/foundation/faq.html (last visited Feb. 13, 2004).

[136]*Id.*

[137]Because the Apache Web server software is open-source, any user can take a copy and run it in modified form. Such a user/developer may modify the software for her needs. If she decides to submit her changes to ASF, she does so in the hope that it will incorporate the changes into the "official" version. The ASF leaders will evaluate the changes and decide whether to accept them. Nikolaus Franke & Eric von Hippel, *Satisfying Heterogeneous User Needs via Innovation Toolkits: The Case of Apache Security Software* (Jan. 2002), *at* http://userinnovation.mit.edu/papers/1.pdf, at 10–11.

[138]This motivation for submitting changes, avoiding back-fitting costs for new versions, can exist for any open-source project. Skilled users will generally have a sense of whether their changes are substantial and important enough, and sufficiently well-implemented, to have a chance for inclusion. If not, there are other ways for the user to automate and minimize the work required to back-fit customized changes into new versions of the open-source software. In the Linux kernel, for example, rather than a group decision as in Apache, Torvalds is intimately involved in the decision whether to accept changes or additions into the kernel. Healy & Schussman, *supra* note 108, at 19–20.

of June 2003, is on approximately sixty-eight percent of active Internet-connected computers.[139]

The Apache open-source license differs from the GPL used in much of Linux[140] in a number of ways. In whole, it is much less restrictive. In essence, it allows one to take the source code and do whatever she likes with it, as long as certain attributions and notices are carried forward with modified or unmodified versions.[141] Thus, the Apache license is almost a full dedication to the public domain, whereas the GPL license is only a partial dedication.[142] The difference is that anyone can use material in the public domain as the basis for a commercial offering with royalty fees for use. The GPL disallows this. The Apache license permits it, and does not even require that the source code be

---

[139]Netcraft, *June 2003 Web Server Survey*, *at* http://news.netcraft.com/archives/2003/06/12/june_2003_web_server_survey.html (last visited Feb. 13, 2004). This survey polled just under forty-one million Web servers on the Internet. Microsoft's various Web server products, in aggregate, had approximately twenty-four percent of the server installations. *Id.* The Microsoft percentage is likely higher if one takes into account internal corporate Web servers providing "intranets" for companies. Such servers are typically behind an Internet firewall and thus are not surveyed by the Netcraft methodology.

[140]Linux, as I use the name here, and in popular use, is really an aggregation of many software components from various groups within the open-source community. Thus, while most of Linux is licensed under the GPL, there are components that are licensed under other open-source licenses. Webbink, *supra* note 8, at 674 ("[M]any open-source software packages, including all of the major Linux distributions, contain several of these licenses, and licensees have not expressed any degree of difficulty dealing with those multiple license schemes."); David Wheeler, *More than a Gigabuck: Estimating GNU/Linux's Size*, *at* http://www.dwheeler.com/sloc (last updated July 29, 2002) (analyzing Red Hat Linux distribution along metric of source lines of code, which, using automated software tools, in essence counts lines of source code for each of various components in Red Hat Linux distribution, and in doing so notes license applied for each component; by lines of code, GPL was specified license for just over fifty percent of source lines of code).

[141]Apache Software Found., *The Apache Software License, Version 1.1*, *at* http://www.apache.org/LICENSE-1.1 (last visited on Feb. 13, 2004) [hereinafter Apache Software Found., *Apache License*].

[142]*See* Bruce Perens, *The Open Source Definition*, *in* OPENSOURCES, *supra* note 2, at 171, 181–83 (describing that GPL does not allow one to take modifications private, but that Apache license is part of family of licenses that allow great deal of freedom with source code and do not restrict privatization).

included with redistributions of the software.[143] The Apache license does require that its almost de minimis conditions be applied to successive redistributions, modified or not.[144]

Apache is only one of several important Internet technologies that rely on open-source software.[145] It is often paired with Linux for Web-server installations. As a result, this tandem occupies an important space in the Internet infrastructure. First, the tandem provides the functionality for a computer to serve content to users on the Web. Second, each piece of the tandem operates in direct competition to offerings from Microsoft. The second point means that these two products are under direct and fierce competitive threat. Their success in light of that threat is significant for the open-source movement.

---

[143]Of course, most users would not pay for an object-code-only copy of Apache when they could obtain the software for free from ASF and rebuild it themselves, unless they lacked the skills to do so or the cost of doing so was more than the redistributor's licensing price. Similar dynamics have kept the Apache project from forking—a theoretically dreaded event for an open-source software project. Forking is when a group takes a copy of a project at a particular stage of development and marches off in a new direction to implement a different vision for the project. The new group may want a different set of features and functions, may want to emphasize different technologies, or simply may be unable to get along with the other developers. *See* Joode, *supra* note 122, at 15, 20–21 (noting that fork can be initiated by developers inside or outside community, and that fork is complete once it develops to point where there are irreconcilable differences between two projects, meaning that it is no longer practically possible, that is, beneficial given costs, to reintegrate two software projects); McGowan, *supra* note 4, at 263–64, 278 (discussing possible reasons for forking, but noting that it rarely occurs). One disadvantage that the new, forked project may have is brand recognition—it may be unable to use the name of the originating software project. Joode, *supra* note 122, at 20 ("The maintainer can only demand that a fork receives a new name, as the original name is likely to be trademarked."). Thus, trademark law may have an anti-forking effect.

[144]The substantial differences between the Apache license and the GPL reflect their history. Stallman wrote the GPL with the specific goal to facilitate the creation of freely available software (with source code) that could not be privatized in certain ways: no royalties for use and the same GPL terms must apply to closely coupled software. Stallman, *supra* note 45, at 59–60. The Apache license, however, derives 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. Joode, *supra* note 122, at 55–56 (discussing pros and cons of practically unrestricted BSD license, in particular criticisms from open-source community that it makes no sense because companies "can use, modify and sell the software and the license does not require them to contribute anything back to the community that originally developed the software."); McKusick, *supra* note 118, at 42–46 (describing initial decision to offer entire BSD Unix flavor under BSD license, due to popularity of networking component earlier offered under license, and discussing later, related lawsuit that pitted other major flavor of Unix, from AT&T, against free BSD Unix distribution, dispute being whether small number of copyrighted AT&T components were present in kernel of BSD Unix).

[145]Wacha, *supra* note 7, at 20 (noting that key Internet open-source programs include "one of the dominant Web programming languages (Perl), the program that routes more than 80 percent of all Internet email messages worldwide (Sendmail), [and] the program that is the basis for the domain name system (BIND)").

Linux and Apache are two of the most successful examples of the open-source approach. They also show that, at least thus far in the development of open-source software, the prominent projects have been oriented to a technical user base.[146] A group of similarly situated technical users are in the best position to take advantage of the collaborative opportunities that the open-source approach offers. These groups need the ongoing access to the source code to successfully improve the software. Their integrity in the work, as programmer-authors, is in the source code and the collaborative work that results from their aggregate effort.

(b) Characteristic Applications for Open-Source Software

The examples of Linux and Apache are both unique and paradigmatic. They are unique due to their prominence, market share, and importance to the Internet. They are paradigmatic in that they characterize the current application space of much open-source software: projects and products for the Internet infrastructure; or for computing technologists, including programmers, and affiliated disciplines.[147] For example, when the district court in the Microsoft antitrust trial made its findings of facts that a large number of available applications supported an operating system's market power, the court considered the possibility that Linux might represent a competitive entrant in the antitrust market that Windows dominated. The court, however, rejected this finding, in part because the applications available for Linux were insignificant compared to applications available under Microsoft's Windows operating system.[148]

While many open-source projects since the Microsoft antitrust trial have developed in non-technologist application spaces, they are a substantial minority of the open-source movement. This raises the question whether the open-source approach is more likely to generate projects and products in certain application spaces, such as platform technologies, but is less likely to

---

[146]One counterexample is the Mozilla project, an open-source Web browser. *See* Mozilla Org., *Mozilla.org at a Glance*, *at* http://www.mozilla.org/mozorg.html (last visited Feb. 13, 2004) (stating that "Mozilla is an open-source Web browser, designed for standards compliance, performance and portability"). Mozilla is an open-source offshoot from Netscape's Internet browser. Netscape dedicated the code to its commercial browser as a strategy to attempt to capitalize on the open-source movement to infuse its browser with energy from the movement. Jim Hamerly et al., *Freeing the Source: The Story of Mozilla*, *in* OPENSOURCES, *supra* note 2, at 197, 197–98, 200–03.

[147]Bessen, *Free Software*, *supra* note 5, at 17 ("It is true that most open-source products are directed at technically sophisticated users, and many are not very 'user friendly.'").

[148]United States v. Microsoft Corp., 84 F. Supp. 2d 9, 23–24, 26 (D.D.C. 1999).

do so in other areas, such as end-user applications.[149] Notwithstanding the present dichotomy and seeming better fit for open-source projects with technologist products, as opposed to end-user products, open-source projects such as Linux, Apache, and others, have broad and important commercial impact.[150]

3. Commercial Mainstreaming

Besides engendering and facilitating programming volunteerism on an enhanced scale, the open-source approach also has tapped and released entrepreneurial and commercial interests that seek to capitalize on the open-source phenomena.[151] The two most well-known examples of this are Red Hat and IBM. The first is a dot-com boom survivor whose business model is based on Linux. The second, a mainstay in computing since its beginning, has joined the Linux bandwagon. IBM's future business model for its computing products

[149]Commentators, especially in economics, are studying open-source software to understand whether the approach is inherently more likely to facilitate projects in certain computing technology submarkets. *See, e.g.*, Raymond, *supra* note 125, § 10 (discussing criteria when open-source approach makes greater economic sense, and contrasting these with criteria that predict that closed-source approach would be better); Klaus M. Schmidt & Monika Schnitzer, *Public Subsidies for Open Source? Some Economic Policy Issues of the Software Market*, *at* http://opensource.mit.edu/papers/schmidtschnitzer.pdf (Nov. 2002), at 14 (discussing competitive disadvantage of open-source development in targeting markets and arguing that it lacks incentive and organizational structure to aggregate market preferences; for software "it is not just the amount of effort and investments in innovation that is important. It is also the direction of technological change and how the innovations respond to what consumers want."). On the other hand, other commentators posit general applicability of the open-source approach, beyond software and source code to information production generally. *See* Benkler, *supra* note 4, at 381–84, 434–36 (postulating open-source inspired peer-production model that is alternative to organizing production according to markets or firms). The degree to whether the open-source approach successfully generalizes into other endeavors is beyond the scope of this Article. The possibility, however, is of interest because endorsement and feasibility of the approach in other information production activities strengthens the approach's hold on source code.

[150]Christopher Wood, *Special Report: Emerging Technologies*, AM. BANKER, Apr. 1, 2003, at A8, *available at* 2003 WL 3344972 (discussing and quoting interview with Eben Moglen) ("Part of what's driving all of this is [that open-source software gives] people an enormous benefit not just because the software is of enormously high quality, and low cost, but it allows migration to low-cost generic hardware, and thus reduces industry dependence on particular vendors.").

[151]John Ousterhout, 42 COMMUNICATIONS OF THE ACM No. 4, at 44, 44–45 (Apr. 1999) (arguing that on its own, "open-source software lacks essential ingredients for mainstream adoption," thus requiring commercial entities to help mainstream software, which additionally provides greater resources for open-source software); *See also* Bulkeley, *supra* note 88 (describing open-source software's move into mainstream business).

and substantial service business revolve around Linux.[152] In essence, IBM has discontinued the business of programming its own flavor of Unix in favor of Linux. Before exploring commercial mainstreaming and its impact on the open-source movement, we must first look ahead to one aspect of the paradigmatic open-source license: the GPL. Examining the GPL will aid our understanding of how companies such as Red Hat and IBM can make money from Linux, a "free" operating system.

Some open-source licenses, including the GPL, specify that the software must be distributed royalty-free, but this does not preclude making money from the open-source software. "Royalty-free" means that an ongoing fee may not be charged for use of the software.[153] However, a distributor may charge for the initial service of providing the open-source software.[154] Red Hat has

---

[152]Int'l Inst. of Infonomics, University of Maastricht & Berlecon Research GmbH, *Free/Libre Open Source Software: Survey and Study: Part II—Firms' Open Source Activities: Motivations and Policy Implications*, at 12 (2002), *at* http://www.berlecon.de/studien/downloads/200207FLOSS_Activities.pdf (last visited June 27, 2003, report overview page available at http://www.infonomics.nl/FLOSS/report) [hereinafter FLOSS Activities] (discussing IBM's $1 billion investment in Linux and its other open-source software activities).

[153]GNU, GPL, *supra* note 99, § 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.").

[154]*Id.* at § 1; Wacha, *supra* note 7, at 22. In large open-source projects, such as Linux, the distribution service price represents value for aggregating and packaging the software.

> Some may ask whether the same [open-source] software for which companies charge money is available free on the Web. The answer is yes and no. While the source code can be reconstituted from several hundred sites (or source trees), even a talented software engineer may not succeed in creating an exact image. Given the way Linux exists on the Internet, . . . it is extremely complex to build. To create the necessary binaries and host environments, a potential user would want to dedicate a substantial team of engineers working every day in the open-source community. Moreover, a user then would need to maintain the code (with more than two million lines in the Linux kernel alone) and update it with new versions of the Linux kernel and patches, among other things.
>
> For most companies, it boils down to a classic make v. buy decision. Even if a potential user could find the same code at no charge somewhere, it is usually more efficient to get that code from expert Linux providers than to develop in-house expertise. Additionally, a potential user who buys code from a vendor often obtains warranties, indemnifications, support, maintenance, upgrades, training, professional services (such as custom software development) by expert developers, and assurances of a reliable, quality-assured binary distribution . . . .

Id. at 23 (citations omitted).

From Wacha's account, it is apparent that there is a market constraint on the price for aggregating and delivering the open-source software. If providers raise prices the market may attract entrants, who can collect the software from the source trees, and regenerate the distribution package for sale.

done precisely this for low-end, Linux-based systems.[155] Red Hat also offers other Linux distribution packages for high-end computing environments, along with increasingly sophisticated, valuable support and affiliated services.[156] Thus, the open-source approach allows one to charge for a software-related distribution and support services even while prohibiting, under some open-source licenses, royalties for use of the software.

The open-source approach enables a number of other business models beyond the distributor model.[157] Red Hat is perhaps the best known example of the distributor model, but there are also other significant Linux distributors.[158] For example, traditional, royalty-licensed software is also a viable business model. This model is typically possible in conjunction with Linux or other open-source software designed as a foundation technology.[159] Linux's function, like any operating system, is to enable other software to run on the computer.[160] The open-source licensing for Linux allows both open-source software and traditional software to execute on the Linux operating system. For example, Oracle is the most ubiquitous commercial database software package in computing. There are many pricing structures under which one can license use of the Oracle database, but Oracle's database is not open-source

---

[155]At one point in time, Red Hat sold a personal copy of Red Hat Linux for approximately forty dollars; for this price, one would receive the media containing Red Hat's Linux distribution, installation tools, thirty days of service, and documentation. Red Hat Linux 9 Web Page, *at* http://linuxstore.se/images/8000.pdf (last visited Feb. 13, 2004). 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.* This signals that Red Hat's primary added value for this forty dollar offering is its aggregation of the software into a distribution package.

[156]*See generally* Red Hat Store, *at* http://www.redhat.com/apps/commerce/ (last visited June 1, 2003) (listing prices for various "enterprise" versions of Linux distribution packages, noting that software is "designed for mission-critical enterprise computing and certified by top enterprise software vendors," and with highest prices running into four figures).

[157]Raymond, *supra* note 125, § 9 (discussing five known and two speculative indirect-sale-value models for open-source software based business).

[158]*See* Michelle Delio, *Linux Distributors Gang Up*, WIRED NEWS (May 30, 2002), *at* http://www.wired.com/news/linux/0,1411,52864,00.html (last visited June 1, 2003) (describing consortium, United Linux, www.unitedlinux.com, among certain Linux distributors and whether other distributors such as Red Hat would join); Major Linux Distributions and Distributors, *at* http://www.linux.org.uk/Distribution.html (last visited June 1, 2003) (listing major distributors for Linux, including Red Hat, which is dominant in United States, SuSE, popular in Europe).

[159]Generally speaking, most open-source software will allow proprietary software to be associated with it in certain ways without upsetting the open-source license terms. For example, practically all open-source licenses allow one to distribute proprietary and open-source software on the same media. For a more detailed discussion of which associations might upset open-source license terms, see *infra* Part III.B.4.

[160]Recall the cook in the kitchen analogy, where the coriander chicken recipe was the software, the cook was the operating system, and the kitchen equipment was the computing hardware. Only the cook (the operating system) could implement a recipe (run a software program) with the kitchen's equipment (the computing hardware).

software.[161] Even so, a user can run Oracle on Linux.[162] Thus, for some traditional software vendors, open-source software may open new markets, provided, however, that an open-source project does not arise and supplant the traditional software vendor's product market.[163] Traditional software is an industry with tremendous variety. It has many vertical markets and technology niches. It remains to be seen which segments of the traditional software industry are amenable to the open-source approach.

A third example of open-source mainstreaming is IBM's Linux embrace. IBM's move, from its own proprietary flavor of Unix, to Linux, implements another business model promoted for the open-source approach: sell complimentary goods and services into the market generated by the open-source foundation technology. In IBM's case, Linux is the foundation technology. IBM is selling its computing hardware and a variety of information technology services. It has traditionally done this; but under a Linux strategy, IBM's efforts not only support a movement that competes with

---

[161]*See* OracleStore, *at* http://oraclestore.oracle.com/OA_HTML/ibeCCtpSctDspRte.jsp?a=b (last visited June 1, 2003) (describing licensing parameters to purchase license to use Oracle software, including licensing by number of computers, processors in computer, or users).

[162]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).

[163]For example, one of the most popular open-source applications is the MySQL database. *See* My SQL, *at* http://www.mysql.com (last visited June 1, 2003) (describing MySQL as "The World's Most Popular Open Source Database"). MySQL is an application like the Oracle database that needs to run on an operating system such as Linux, and which provides data management capability. The acronym "SQL" stands for structured query language, a programming language (of a certain sort) to manipulate data in a database. From an uninitiated view, MySQL is in the same market as the Oracle software, since they are both databases. However, there is currently minimal threat of actual significant competition despite the apparent similarity because Oracle is targeted and designed for applications of scope and complexity beyond MySQL's capabilities. The two products are in different classes of the database market. This may not always be the case. One of the questions for open-source software is whether in the future it will spawn projects in the application product space with the impact and prominence of its applications in operating systems and Internet infrastructure. *See* Douglas C. Schmidt & Adam Porter, *Leveraging Open-Source Communities To Improve the Quality & Performance of Open-Source Software*, 2–3, *at* http://opensource.ucc.ie/icse2001/schmidt.pdf (last visited June 1, 2003) (position paper submitted for ACM Workshop: Making Sense of the Bazaar: 1st Workshop on Open Source Software Engineering) (describing application types, or "domains," where authors question whether open-source approach will be successful, including niche and vertical markets, low-margin markets, and secure computing markets). *See also* Raymond, *supra* note 125, § 10 (discussing when company should strategically open-source its product to preempt competitors, be first mover to recruit best community of developers, and transition its business model from direct rents by software sale to leveraged business model of complimentary goods and services).

its rival, Microsoft, but also provide other competitive advantages.[164] IBM's invested resources in Linux, combined with its prominence in computing, are a notable tipping point for Linux and the open-source approach. They signal large-scale viability for open-source, or at least for Linux. With IBM behind it, Linux's credibility is enhanced, making large organizations more likely to choose it for their information technology needs. IBM has a mix of product and service offerings so that even if it no longer makes license royalties from its flavor of Unix, its overall business prospects are brighter because it can ride the growth of Linux for a competitive advantage and additional non-operating-system business.[165]

These three examples—distribution, traditionally licensed software, and complementary goods and services—demonstrate the commercial mainstreaming of the open-source approach. It is an ongoing phenomenon; to what degree mainstreaming will occur across a greater variety of markets and applications is a question about which there are many predictions and much study. The movement from an original ideology, ensuring the freedom to view and share source code, to an approach of developing and licensing software that supports new approaches to business, parallels a similar progression in the Internet: the original notion of a communications network, to support research, enabled the development of a richer network experience via HTTP and browser technology, which in turn laid an irresistible foundation for commerce over the Internet.[166] Commercial mainstreaming says several things about the open-source approach. It signals value in the approach, both in the opportunities created for new business models and business models recycled to fit open-source. It foreshadows conflict over the approach as the stakes climb higher and commercial opportunities fit themselves to the approach,

---

[164]It is reported that IBM has invested over $1 billion into Linux-related development. *See* FLOSS Activities, *supra* note 152, at 7. *See also* Raymond, *supra* note 125, §§ 9.2, 10 (calling hardware vendor's business model "widget frosting" to emphasize that open-source software makes vendor's equipment more attractive and tasty).

[165]Raymond, *supra* note 125, at 17–18, 21.

[166]*See* CODE, *supra* note 44, at 30, 39 (describing growth of Internet from research tool to commercial tool).

amplifying it.[167] Products such as Linux would not have grown as they have without commercial entities to facilitate delivery of the open-source software to organizations that wish to deploy it without aggregating it themselves.

Commercial mainstreaming points back to the open-source license—a conditioned permission to use, modify, and distribute the software along with the source code.[168] The conditions are unique in many ways when compared to traditional software licensing. Large scale commercial use of open-source software puts greater scrutiny on the various open-source licenses and on the open-source approach in general. I have sketched the open-source approach in a prior Section, but a more detailed look is necessary for the rest of my argument. Thus, the next Section reviews the key attributes of open-source software licenses, contrasting the differences between some of the more prominent licenses.

---

[167]As a prominent computer company, IBM's embrace of Linux has engendered a lawsuit by a small software company allegedly holding certain rights to a flavor of Unix, called AIX, that IBM has licensed, developed, and distributed for many years. Steve Lohr, *No Concession From I.B.M. In Linux Fight*, N.Y. TIMES, June 14, 2003, at C1 ("The case, regardless of its outcome, also points to a broader issue that will not go away: how to manage the meeting of two worlds of programming."). The plaintiff's claim is that IBM has transposed plaintiff-owned AIX software into Linux. *Id.* The suit has generated much commentary in the open-source community because IBM's embrace of Linux was a major benefit for Linux. *Id*. *See* John C. Dvorak, *Killing Linux; Linux and the whole open-source movement are in peril*, PC MAG., June 1, 2003, §§ 1–3, *available at* 2003 WL 5729467 (arguing that open-source community is not taking suit seriously enough, nor addressing underlying risk of open-source programming—contributor who, unbeknownst to rest of group, injects protected code into project). *See also* Eben Moglen, *Enforcing the GPL I*, § 6 (2001), *at* http://emoglen.law.columbia.edu/publications/lu-12.html (last visited June 26, 2003) (discussing how GPL licenses open-source software); Eben Moglen, *Enforcing the GPL II*, §§ 2–6 (2001), *at* http://emoglen.law.columbia.edu/publications/lu-13.html (last visited June 26, 2003) (discussing enforcement efforts for open-source software under GPL).

[168]My earlier review of Linux and Apache described two ends of a licensing continuum. Comparatively, Linux's GPL puts a great deal of "control" over users and redistributors, but it is control in the sense of ensuring openness. On the other hand, the Apache license very minimally restricts the software. Redistributors are free to distribute only object code and charge royalties, so long as certain attributions and notices are given. An expanding number of open-source licenses sit somewhere in between these continuum boundaries. Because licenses have proliferated, leaders in the open-source movement established the Open Source Definition ("OSD"). Raymond, *supra* note 125, § 8 ("[T]he Open Source Definition . . . was written to express the consensus of the hacker community regarding the critical features of the standard licenses . . . ."). *See* Open Source Initiative, *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."). The OSD is a set of guidelines. OSI evaluates licenses against the guidelines to determine whether a license meets the OSD, publishing the evaluations on its Web site. *See* The Approved Licenses, *at* http://opensource.org/licenses/index.html (last visited June 4, 2003) (listing many dozen approved licenses).

B. Open-Source Software Licenses and Collaborative Development

In this Section I focus on open-source *software* licenses, but the picture would be incomplete without first relating how the open-source approach has sprouted in other copyrightable subject matter. Software is a unique type of copyrightable material due to its dual character: it is both a functional writing and an expressive work. The functional nature heightens the importance of access to the source code. Thus, source code availability is a key provision in open-source software licenses. Non-software copyrightable subject matter can dispense with this condition in a license to promote sharing. However, the remaining open-source conditions have inspired their application beyond software.

1. Open-Source Licenses for Non-Software Subject Matter

Certain undertakings seek to extend the open-source approach to information generally, or at least to non-software copyrightable material. One prominent example is the Massachusetts Institute of Technology's ("MIT") Open Course Ware project, which is "open-sourced" under the Creative Commons.

The MIT project seeks to "[p]rovide free, searchable, access to MIT's course materials for educators, students, and self-learners around the world . . . [in an] efficient, standards-based model that other institutions may emulate to openly share and publish their own course materials."[169] It does this under a licensing approach promoted by the Creative Commons, an organization started by a group of activists, including cyber-law and intellectual property experts.[170] 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."[171] Content producers can select from a menu of possible licenses, each specifying a

---

[169]OCW, *About OCW*, *at* http://ocw.mit.edu/OcwWeb/Global/AboutOCW/about-ocw.htm (last visited Feb. 18, 2004).

[170]Creative Commons, Frequently Asked Questions, *at* http://creativecommons.org/faq (last visited June 3, 2003) [hereinafter CC-FAQ]. *See generally* Thomas F. Cotter, *Prolegomenon to a Memetic Theory of Copyright: Comments on Lawrence Lessig's The Creative Commons*, 55 FLA. L. REV. 779, 780–85 (2003) (discussing "emerging field known as memetics" and its implications for copyright law); Molly Shaffer Van Houweling, *Cultivating Open Information Platforms: A Land Trust Model*, 1 J. TELECOMM. & HIGH TECH. L. 309, 319–323 (2002) (analogizing open-source license to land trusts); Maureen Ryan, *Cyberspace as Public Space: A Public Trust Paradigm for Copyright in a Digital World*, 79 OR. L. REV. 647, 648 (2000) (suggesting "a public trust paradigm for formulating copyright principles in a digital world").

[171]CC-FAQ, *supra* note 170. 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.*

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.

2. Pre-Open-Source Sharing (Typically) Without Source Code

Creative Commons and examples like it show that the open-source approach may apply beneficially beyond software. However, this extension emphasizes the unique condition necessary for open-source software: availability of the source code. After all, before the open-source approach rose to prominence, there was an active trade in freeware and shareware software.[172] These programs were usually developed and written in a similar vein as single-programmer open-source software. Hobbyists and tinkering programmers built interesting or useful software and then offered it, typically in object code form, to others free of charge, or in the case of shareware, for a de minimis fee.

The freeware and shareware history demonstrates a sharing tradition among programmers, in particular for utilities, tools, and components useful to the tasks of developing software, integrating software and systems, or operating information technology systems. Writing and sharing these programs serves some of the same functions for programmers as developing open-source software serves: it creates an opportunity to experiment and learn; it provides a solution to a computing problem or preference that has been annoying or troubling the programmer; it allows the programmer to establish a reputation by sharing her solution with others; it lets the programmer signal her characteristics as generous, clever, talented, or skilled with a particular technology; and it allows the programmer to promote the project as a signal of her skill, such as by listing the freeware or shareware project on a resume, or

[172]*See, e.g.*, Int'l Inst. of Infonomics, University of Maastricht & Berlecon Research GmbH, *Free/Libre and Open Source Software: Survey and Study: Part III—Basics of Open Source Software Markets and Business Models*, at 11–12 (2002), *at* http://www.berlecon.de/studien/downloads/200207FLOSS_Basics.pdf (last visited June 27, 2003, report overview page available at http://www.infonomics.nl/FLOSS/report/) [hereinafter FLOSS Basics] (presenting two-by-two classification scheme for software using axes of source code availability and royalties, resulting in following four categories: shareware/freeware, commercial open-source software, noncommercial open-source software, and proprietary/commercial software); Schmidt & Schnitzer, *supra* note 149, at 4 (distinguishing open-source software from freeware and shareware).

bringing a copy to demonstrate at an interview.[173] However, what the freeware and shareware practice generally lacked was collaboration on any meaningful scale. The open-source approach does not mandate that collaboration, but it permanently facilitates it. Most open-source software licenses ensure to varying degrees that during the entire life of the project it is always susceptible to collaborative energy and input. The next Subsection will discuss how.

[173]Under the traditional software development model, programmers may be unable to show any direct work product to potential employers during an interview. This is so because the prior employers likely extracted a contractual promise that the programmer hold confidential any software developed and contractually designated the effort as a "work made for hire" under copyright law. *See* Cmty. for Creative Non-Violence v. Reid, 490 U.S. 730, 737, 739–41 (1989) (holding that terms "employee" and "scope of employment" follow common law of agency for purposes of copyright work for hire doctrine, and noting that "contours of the work for hire doctrine therefore carry profound significance for freelance creators—including artists, writers, photographers, designers, composers, and computer programmers . . ."); Rochelle Cooper Dreyfuss, *The Creative Employee and the Copyright Act of 1976*, 54 U. CHI. L. REV. 590, 594–97 (1987) (noting history and importance of work for hire doctrine and discussing factors courts use to determine whether work is "work made for hire," thus vesting original copyright authorship with employer or commissioning entity). Further, if an employed programmer takes a copy of the source code she developed, this may create a risk of trade secret violation and breach of the programmer's employment agreement. Many programmers create software for internal use by their employers. Raymond, *supra* note 125, § 6. Such a programmer would perhaps be unable to show future employers either the source code she has written or her applications as they function when running, at least without running the risks of unauthorized disclosure. An alternative for these programmers is to write freeware or shareware. This software is totally under their control. Thus, the programmer can hold it up as a credential when exploring opportunities beyond her present employment. An open-source software project would provide the programmer with similar benefits, enhanced by the possibility that the programmer may be able to emphasize to potential employers her status and role within the contributing group that is developing the open-source software.

3. The Open-Source Approach in a Collaborative Software Project

An open-source software project with multiple contributors over time generates a web of license permissions that in all likelihood locks the project into open-source status for its useful life.[174] Each contributor to an open-source software project adds subject matter potentially protected by copyright.[175] The result is that for each contributor to use and further modify the software, she relies on the permission granted in the open-source license by all other contributors. Recall Allen, Betty, and Carol, and their open-source software, GoneOutdoors. If only these three contribute, the web of permissions is small. But assume that over several years, twenty users, who are also programmers, add significant new functionality and send the changes back to Carol. She incorporates the changes into GoneOutdoors and regularly republishes the new

[174]*See cf.* McGowan, *supra* note 4, at 259 ("This Web of blocking copyrights suggests that, as a practical matter, each contributing programmer would have to agree to privatize the code if it was to be taken private in its most current and complete form."); *id.* at 264, n.116 (describing bars preventing single author from privatizing code). *See also* Mark A. Lemley, *The Economics of Improvement in Intellectual Property Law*, 75 TEX. L. REV. 989, 992–93, 1073–77 (1997) (arguing that copyright law should change to allow improvements similar to those that occur in patent law, where improver can own patent rights in new technology even if that technology infringes preexisting patent). In the open-source approach, the web of blocking copyrights results from the open-source license. Permission has been granted to the improvers to make modifications to the software. Typically, most members of the project team will want to use the latest version of the software incorporating all team members' changes. As a result, by so using, they need to rely on the open-source license, which generates the web of copyright permissions. The situation is further complicated by considering non-developer users. If all contributing programmers privatize the open-source software, which would allow them to take the software in a number of traditional directions for licensing and distribution, the group would have to live with the existing user base's access to the source code. Under most open-source licenses, these users would have the permission to continue to develop the software. They would have no right to the privatizing group's future changes and resulting (presumably better) versions; but they could continue an open-source vein of the software.

[175]As a literary work with "thin" copyright protection, not all elements of the source code are protectable. Menell, *supra* note 2, 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.").

version on her Web site. Allen, Betty, and Carol have, in effect, allowed twenty others to join them as co-contributors[176] to GoneOutdoors.

As the developing group grows in a project with multiple versions over time, so grows the interdependence of license permissions. This is in part because the latest version is typically the most desired version of the software. Assuming that the software has added features and functionality, the latest version most likely has contributions from the largest group.[177] Alternatively, although it is possible that later versions could excise all of the code contributed by one or more programmers in favor of new code from a new group, this is not usually the case when the software obtains increased capability with each version.

The interdependent web of license permissions resulting from the open-source approach defines only a bare minimum in "ground rules" for open-source software development.[178] Compared to traditional software development processes, much is left unspecified. It is not the purpose of open-source software licenses to specify the pragmatic details about how the contributors will coordinate their collaboration; but the collaboration would be perhaps impossible, or at least exposed and vulnerable, without the open-source approach. The group relies on both the norms the open-source license enforce and the traditions it expresses.

Whether collaboration occurs, and its effectiveness, depends on factors that define the software development process. These characteristics are best

---

[176]By designating the programmers as co-contributors, I consciously sidestep the question of whether the co-contributors are joint authors in the copyright sense, or whether the resulting product is a joint work. *See* NIMMER & NIMMER, *supra* note 55, § 6.03 (discussing elements of joint authorship and distinction between joint authors and joint work). While a determination of joint ownership among the contributors to an open-source project would upset the open-source licensing scheme, this risk explains in part the need for a license that asserts ownership in the original author for her contribution, and then grants conditional rights to others.

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. § 6.04 (citations omitted).

[177]Raymond, *supra* note 125, §§ 5, 11.

[178]Bruce Kogut & Anca Metiu, *Open Source Software Development and Distributed Information*, 17 OXFORD REV. OF ECON. POL'Y 2, 248, 257 (2001).

understood in two contexts: first, by their manifestation in traditional software development, and second, in open-source software development.

(a) Traditional Coordination of Team-Developed Software

Traditional software development is command and control, sometimes to a severe degree. Its characteristics are resource measurement, progress tracking, product and customer requirements, work hierarchy, code partitioning, and a variety of technological and practical constraints on the implementation options available to a programmer in the project.[179] All these characteristics are in the forefront of team-developed software.

For example, programmers in traditional software development teams may find themselves forced to work with old technology by their employers.[180] The interests of the programmers and the company diverge because the costs to convert the software do not exceed the gain to the company of doing so.[181] This outcome results because the command-and-control hierarchy of traditional software development is often styled after traditional industrial production methods.[182] This approach is in part a response to the corporate need to predictably guarantee and generate outputs when expending production inputs.

However, the history of traditional software development shows that the industrial production method to produce software has limits. Software design and programming are a unique mix of science, engineering, and practitioner art. One example of this mix manifests in a dramatic degree of variability

---

[179]For a discussion of the traditional software development process to develop and formalize project requirements, see Walt Scacchi, *Understanding the Requirements for Developing Open Source Software Systems*, 2, 7–8 (Dec. 2001), *available at* http://www.ics.uci.edu/~wscacchi/Papers/New/Understanding-OS-Requirements.pdf (last visited June 13, 2003) ("The focus in this paper is directed at understanding the requirements for open software development efforts, and how the development of these requirements differs from those traditional to software engineering and requirements engineering . . . .").

[180]The programmers may work on a software product that is mature, yet has a large user base demanding some new functionality. If the product is written with an "old" programming language, it may not be worth the company's investment to convert (reprogram) the software to a modern language, even if the programmers prefer this to update their skills.

[181]This cost benefit analysis may or may not hold true, and is determined in part by the management philosophy of the company in how it measures short-term versus long-term gains. Converting the software may not generate a positive return for the life of the project, meaning that the up-front cost is greater than the present value increase in rents the company will obtain from the conversion, taking into account increased revenues, cost structure changes, and post-conversion profit levels. If the intra-project costs exceed the benefits, alternative extra-project justifications for converting would include the training effect on the programming team, increasing their value for future projects on different software, and increasing the likelihood of retaining them as employees. It is common for programmers to switch jobs in order to get on a project that allows them to upgrade their skills.

[182]*See, e.g.*, Raymond, *supra* note 125, § 3 ("[S]oftware is largely a service industry operating under the persistent but unfounded delusion that it is a manufacturing industry.").

among programmer output. Even measuring this output is fraught with uncertainty. Another example is that despite a history attempting to apply various methods to improve software quality, defect rates are generally perceived as too high and beyond the rates found in other industries.

The management hierarchy in traditional software development provides control and coordination. Larger projects require subdivisions among both the programming teams and the source code.[183] Thus, the source code itself reflects the nature of the hierarchy.[184] Traditional organizational lines of control are common. Small groups program components of the software. Each group has a supervisor, all of whom may report to the project leader. Often design, programming, and testing are separate functions with specialized or only marginally integrated groups performing each function.[185]

The sketch I have drawn of the traditional software development process is by definition a stereotype—specific programming projects exhibit more or less of these characteristics. By my sketch I do not mean to imply that the traditional methods are obsolete. And the differences between the traditional methods and the open-source development process are ones of degree.[186] In terms of coordination and control among the contributing team, open-source software is innovative, but there is debate as to whether it is truly a new paradigm for software development.

---

[183]Kogut, *supra* note 178, at 258–59.

[184]In many aspects of the software development process, open-source or otherwise, software tools enable and support development. One commonly used tool is a source code control system ("SCCS"). A variety of SCCS software products are available. They typically provide the following capabilities: (1) SCCS systems store all the source code in a common repository and programmers access the code through the SCCS; (2) programmers "check-out" the code from the SCCS to work on it, and when finished, "check-in" the code; (3) the SCCS tracks the changes made by individual programmers; (4) the SCCS manages versioning of the software, allowing programmers to work on specific versions, and in some cases the SCCS propagates changes across versions when appropriate; (5) the SCCS provides automated compilation of multiple source code components into a finished "build" of the product; and (6) the SCCS provides reports and other tracking tools for all of its capabilities. Part of the design of a large software project is to determine granularity in subdividing the source code into components, such as specific files, or, in more modern programming languages, objects or object classes. The SCCS is a key facilitating tool to implement and manage the organizationally imposed coordination necessary in a large, multi-contributor software project.

[185]*See* Paul Vixie, *Software Engineering*, *in* OPENSOURCES, *supra* note 2, at 93–96 (describing integration and testing process in software engineering).

[186]*See generally id.* at 96–100 (describing open software engineering process). Indeed, some open-source software development techniques are working their way into the corporate development setting. *See* Jamie Dinkelacker & Pankaj K. Garg, *Corporate Source: Applying Open Source Concepts to a Corporate Environment, Position Paper* 2 (2001), *available at* http://www.hpl.hp.com/techreports/2001/HPL-2001-135.pdf (describing application of "Open Source concepts, perspectives and methodologies within the corporate environment," including its debugging success, "best summed up by Eric Raymond: 'given enough eyeballs, all bugs are shallow'").

(b) Coordination of Open-Source Collaboration

In large, multi-contributor projects, open-source programmers must, and do, collaborate, but by means different in degree than traditional software development. Organizationally, the means are informal rather then formal. Most of the controlling structures of traditional software development are relaxed to a degree where they are no longer conspicuous. Technologically, however, tools similar to those used in traditional software development are employed to partition, segment, and manage contributors' inputs to the project.[187]

The key differentiating characteristics of open-source software development are: (1) self-identification for roles and tasks; (2) geographically disperse development groups; and (3) partial merger of the sometimes traditionally separate design, programming, and support functions. Self-identification is inherent in open-source development. The programmer who initially starts a collaborative project (as opposed to a sole effort) is often self-identifying to act as the leader or facilitator of the project, at least initially. Programmers who volunteer to contribute to the project typically identify the improvements they would like to program, or at least identify an area or aspect of the software in which they would like to work and for which they are well suited.[188] This approach is rational because, among the many possible motivations that drive programmers to contribute to a project, one of the most commonly recognized is the desire to establish or enhance their reputation.[189]

---

[187]The most popular SCCS used for open-source software projects is called the Concurrent Versions System ("CVS"). *See* Online Chapters from KARL FOGEL, OPEN SOURCE DEVELOPMENT WITH CVS, *at* http://cvsbook.red-bean.com/cvsbook.html (last visited June 26, 2003) (providing "a set of free, online chapters about using CVS (Concurrent Versions System) for collaboration and version control"). CVS is an open-source software product. *Concurrent Versions System: The open standard for version control*, *at* http://www.cvshome.org/ (last visited June 26, 2003). *See also* von Hippel, *supra* note 123, at 219 ("CVS is an important software tool used by many open-source projects.").

[188]Commentators have posited that self-identifying for tasks in the open-source project is an important reason why the open-source approach purportedly has produced higher quality software. The thesis rests on the nature of software development—it is often as much art as science because creativity is required to solve many computing problems. Another condition of this thesis is that managing human capital for creative endeavors such as programming is difficult, because it is inherently difficult to match the task's requirements with the programmer's creative strengths. Under the assumption that programmers self-identify to program solutions in areas where they are creatively strong and skilled, self-identification solves this matching problem. *See* Benkler, *supra* note 4, at 375–76, 381, 399 (describing advantages and implications of self-identification).

[189]Eric S. Raymond discusses several aspects of reputation-enhancing behavior, contrasting reputational gains for the prospects of economic reward with reputational gains for social status within the open-source "hacker" gift culture. Eric S. Raymond, *Homesteading the Noosphere*, § 2 (ver. 3.0, 2000), *at* http://catb.org/~esr/writings/homesteading/homesteading (XTML version, last visited Feb. 17, 2004).

Self-identification also may explain, at least in part, the success open-source developers have achieved in coordinating projects remotely through electronic communications. When programmers self-match their strengths and skills to an aspect of the project, they are less likely to need input and direction from other members of the group. By successfully volunteering and contributing in the first place, the contributor is signaling familiarity with the application's technology and the source code in which it is implemented, as well as the capability to communicate in the project's technological lexicon. This relates to and supports the second characteristic of collaborative open-source software: geographically disperse development groups.[190]

The third characteristic of many collaborative open-source projects, in contrast to much traditional software, is the partial merger of design, programming, and support. This occurs because collaborative open-source projects typically have a "core" group of contributors and a secondary group with lesser-scope involvement.[191] Often the core group contributes a supermajority of the code while the secondary group's contributions are less significant both in volume and importance.[192] This same core group, however, has typically made the critical internal and external design decisions. External design questions arise from identifying and characterizing the user requirements of the software. Internal design questions relate to evaluating and choosing among options to implement the external user requirements. The two are interrelated. For open-source software, which traditionally has had sophisticated technologists as end users, the contributors are typically users. This conjunction inherently merges the design and programming process.

Support merges as well into many collaborative open-source projects because, except in rare cases such as the Linux aggregator-distributor, who provides support, the programming contributors typically respond to user queries and bug submittals. This differs from traditional software where there is usually a separate group that interfaces between the end users and the programming team. This is thought to promote a better distribution of resources by specializing the respective functions. Open-source software, by its very nature, lacks the capacity to establish a formal customer support group, so this function is informally distributed among the programmers.[193] Open-source programmers find this less onerous than their traditional counterparts might because the users submitting questions and bugs are typically sophisticated technologists and thus, normally, the submittals are, in a sense, additional

---

[190]Scacchi, *supra* note 179, at 3.

[191]Mockus et al., *supra* note 4, at 323–24, 339–41, 344 ("[M]embers must be persistent and very capable to achieve core status.").

[192]*See, e.g.*, von Hippel, *supra* note 123, at 211 (noting that core group of only eight users generated the initial version of collaborative software that would grow to become Apache).

[193]Mockus et al., *supra* note 4, at 322 ("[P]articipation of the wider development community is more significant in defect repair than in the development of new functionality.").

contributions that improve the software.[194] In addition, open-source software users typically have another resource when they need input and guidance: skilled users who are willing to help answer other users' questions.[195] This has the effect of supplementing the programming group's provision of support while facilitating the growth of the open-source software product, which indirectly benefits all users by extending the network economies of the product.

While these three characteristics—task self-identification, geographic diversity, and merger of functions—differentiate open-source software collaborative development from traditional team-based software development, the animating force is that open-source programmers are motivated to collaborate with an intensity beyond what is present in the traditional environment. Commentators and open-source programmers alike have discussed a variety of hypotheses as to what motivates participants in open-source software projects. The motivation questions arise for both the individual programmer who volunteers time during evenings and weekends, as well as for

[194]The form of the communications may also make a difference. Traditional software support groups typically have obligations to interact with users via telephone calls and perhaps even through site visits. Some traditional software companies offer "Internet-only" support via email, discussions lists, Web sites, and similar technologies. However, these mechanisms are usually the only venue for open-source software support (again, exempting the case of Linux aggregator-distributor companies such as Red Hat). *See* Eric von Hippel, *Horizontal Innovation Networks—by and for users* 5 (MIT Sloan School of Management, Working Paper No. 4366-02, June 2002), *available at* http://web.mit.edu/evhippel/www/UserNetworksWP.pdf (last visited Nov. 27, 2002) ("Most users of open-source software simply 'use the code,' relying on interested volunteers to write new code, debug others' code, answer requests for help posted on Internet help sites, and help coordinate the project.").

[195]User to user support is an interesting sidelight to the provision of design, programming, and support in an open-source software project. *See* Karim R. Lakhani & Eric von Hippel, *How Open Source Software Works: "Free" Under User-To-User Assistance?* 4 (MIT Sloan School of Management, Working Paper No. 4117-00, May 2000), *available at* http://papers.ssrn.com/sol3/papers.cfm?abstract_id=290305 (last visited June 14, 2003) (discussing theories why "some product users voluntarily provide answers to the questions of other users . . ."). *See also id.* at 34–35 (discussing general implications for user-to-user innovation systems).

an entity that authorizes employees to contribute during working hours.[196] An intermediate position for companies is to tolerate employees who contribute during off-hours.[197] Most of the motivational theories support the implication that if programmers volunteer to contribute to the project, they will collaborate well. To obtain the benefits of volunteering, they must facilitate and enable their part of the collaboration.

Although categorization could occur along many lines, I divide the motivational theories for volunteering programmers into the following rough categories: (1) reputation-based theories; (2) career-planning-based theories; (3) rebellion or antiestablishment theories; (4) learning-based theories; and (5) narrow-utilitarian theories, meaning that the programmer is working directly for the benefit of having and using the software or improved software.[198] An example of the first is an account of the open-source community as a gift culture, where programmers value sharing and developing

---

[196]Entities that pay employees to contribute to open-source software may do so under several models. The first model is institutional philanthropy of some sort. The second model is a complementary goods and services business model. Examples include IBM's contributions to Linux, and perhaps Netscape's dedication of its browser code to establish the Mozilla project. *See* Mozilla Org., *supra* note 146 (explaining purpose and workings of Mozilla). The latter example may be philanthropic in part, but also had commercial motives—Netscape dedicated the Mozilla code at a time when it was fighting a (losing) battle with Microsoft's Internet Explorer browser. Also, a third explanation probably present in the Mozilla dedication is marketing and corporate image building. Finally, research institutions and governmental entities may pay programmers to develop software that is later dedicated to open-source status. This is a point of controversy. *See* Hahn, *supra* note 9, at 10 (noting differing views on government's funding for GPL-licensed research). Some argue that federally funded research should aim to develop job-creating commercially viable technology. The argument is that open-source software underutilizes research funds because it will not support a profit-making startup or other commercial endeavor, at least not directly or in the vein of the traditional university technology spin-off company. Evans, *Preferring OSS*, *supra* note 129, at 43, 46–47.

[197]A technology employer might have a competitive disadvantage if it enacted a policy prohibiting its employees from contributing to open-source projects, because many contributors use the open-source project to create a more satisfying mix of programming activity. The employer's power to do so is in part indirect, resting in the various promises contained in the typical high-technology employment agreement. When the employee promises to keep trade secrets and respect the employer's intellectual property, this can create a tension when the employee also contributes to open-source projects in the same or neighboring areas. The opposite effect, however, is possible, where instead of producing tension, a company's open-source policy endears employees. Some technology employers promote that they encourage their employees to contribute to open-source software. A similar effect exists in the law firm labor market for new attorneys where claims of strong pro bono programs are generally perceived as a positive attribute for the employing law firm.

[198]*See* Raymond, *supra* note 189, § 2 (describing various attitudes toward open-source software that approximate given categories). While my last category rings of classic economic motivations, other categories in the list could be analyzed from a broad utilitarian or economic framework.

a reputation for sharing.[199] In this account, this reputation is obtained not so much for its value to future career opportunities, but because it provides the programmer esteem in the community. An example of the second is an account of open-source software participation for career concerns.[200] Here, the programmer participates in order to develop skills and knowledge that reflect positively to future employees. Microsoft abhorrence is an example of the third. Many open-source programmers and leaders in the various circles of the open-source community posit that the movement's energizing, volunteer-capturing force is a fighting response against the power and control of proprietary, traditional software.[201] The fourth classification is self-explanatory: programmers develop open-source software as an alternative to other forms of skills training or professional or intellectual enlargement.[202] Similarly, the fifth category identifies situations where programmers directly want the benefit of the new code they program.[203]

Putting aside a sole-programmer open-source project where collaboration is not relevant, these motivational categories all suggest that contributing programmers will collaborate well. To accrue reputational or career-planning benefits, it helps the programmers if others perceive their work as timely, thorough, terse, and in tune with the project. If the open-source software is to achieve antiestablishment goals and provide an immediately useful output, it must perform well, be a good fit for the application or problem, and attract a user base. If the programmer's goal is self-education, perhaps as a "junior" contributor, collaborating well is essential to partake in learning-benefits from the project. Under all of these motivating forces, contributing programmers

---

[199]*Id.* § 6 ("In gift cultures, social status is determined not by what you control but by *what you give away*.").

[200]*See, e.g.*, Lerner & Tirole, *Simple Economics*, *supra* note 88, at 3 (noting that "labor economics" and "career concerns" can "explain many features of open-source projects").

[201]*See generally* Steven Weber, *The Political Economy of Open Source Software* (BRIE Working Paper No. 140, June 2000), *available at* http://e-conomy.berkeley.edu/publications/wp/wp140.pdf (last visited June 15, 2003) (describing generally politics of open-source software).

[202]*See* von Hippel, *supra* note 123, at 211 ("Today, an open-source software development project is typically initiated by an individual or a small group with an idea for something interesting they themselves want for an intellectual or personal or business reason.").

[203]Justin Pappas Johnson, *Economics of Open Source Software*, § 9 (2001), *available at* http://opensource.mit.edu/papers/johnsonopensource.pdf (last visited Nov. 27, 2002).

could be expected to communicate among the group in ways that facilitate collaborative effort.[204]

This Section began with the proposition that the open-source approach established foundation expectations and norms to facilitate collaboration. It protected the project from opportunists who would co-opt it. But the approach did not specify the means to coordinate the collaboration. Other motivational and technological factors provide the means. The motivations contrast with those of traditional team software development, but the coordinating technology in open-source development builds on traditional methods with the newfound connectivity and bandwidth of the Internet. While the open-source licensing approach primarily sets the stage and protects it from pillage, some of its provisions also influence the coordinating means.[205] The next Subsection elaborates on both roles.

4. Open-Source Licenses and Their Impact on Collaboration

Without the open-source approach, or some other agreement or controlling force among the programmers, the group risks opportunistic behavior by a subset that would appropriate the software for its own purposes.[206] Limiting this collaboration-defeating strategic behavior is a fundamental purpose of the open-source software license. Indeed, such a limit is inherent in the philosophy underlying open-source software—that sharing is the modus operandi. Thus, the open-source license establishes an eco-culture for collaboration. Using a different metaphor, part of the "bargain" individually volunteering open-source programmers expect when contributing to a project is that no one will privatize the project for his or her personal gain.[207] Because software is non-rivalrous and the costs to copy it are small

---

[204]While the motivations for open-source developers may be substantially, or drastically, different from that of traditional software development team members, the means of communication and coordination are technologically similar. Electronic communications, primarily email, listserves, and similar mechanisms, provide transparency and a shared community history. As in traditional development, SCCS software provides technological coordination. SCCS partitions the code and enforces modularity as necessary to enable multiple contributors to coordinate changes to the source code text without blocking each other or overwriting each other's work.

[205]McGowan, *supra* note 4, at 245 ("The licenses that enforce the property rights on which this structure rests are important to its success. The licenses provide a mechanism for enforcing norms, for distinguishing the open-source community from conventional software production and, in some cases, for providing incentives to programmers who require them.").

[206]*See, e.g.*, Lerner & Tirole, *Scope of Licensing*, *supra* note 98, at 4–6, 12–13 (discussing restrictive licenses and "hijacking"); Siobban O'Mahony, *Guarding the Commons: How Community Managed Software Projects Protect their Work*, 5–6, 8–10 (2003), *available at* http://opensource.mit.edu/papers/rp-omahony.pdf (last visited June 13, 2003).

[207]*See* Bessen, *supra* note 5, at 13 (noting that open-source developers want their work to benefit the community).

once one has access to the source code, it has traditionally been viewed as highly susceptible to appropriation. Thus, the open-source approach, at least for a license that prohibits royalties, solves two problems. First, it ensures that the source code is available for collaboration. Second, it removes a disincentive to contribute to a project: the fear that someone undeserving will co-opt the benefit of the group's efforts. This not only facilitates volunteerism, but it facilitates collaboration because group members do not have to police each other to the same degree. They all know that the license binds them to a common course, which reduces interpersonal tension and promotes a sense of mission that traditional software development may find enviable.[208]

(a) Diverging License Terms in Open-Source Software

To further see the impact of the open-source approach on collaboration next requires dissecting the approach. Prominent open-source licenses diverge on certain terms important to collaboration. These diverging key terms are: (1) whether redistributors must provide source code; (2) whether redistributors are allowed to charge royalties for software use; (3) whether, or to what degree the open-source license terms apply to other associated software; and (4) whether redistributors must apply the same terms to their licensees. The two prominent open-source licenses already mentioned diverge on the first two points. The GPL, used for Linux, requires source code with redistributions and prohibits royalties, but the Apache license does not.[209]

The third term, the reach of the open-source license provisions to associated software, is both a difficult technical and legal issue. Among open-source software licenses, the GPL embodies maximum extension of its terms to software associated with software covered by a GPL. The issue is what "associated" means in this context. Some find the GPL's reach expansive, which is why it is sometimes described as "viral" in this regard.[210] "Viral" is not used in the sense of a computer virus, but in the sense that the GPL license terms seek to "infect" the whole of the software that contains any GPL-

[208]*See* Raymond, *supra* note 189, § 19 (arguing that open-source programmers are more motivated than their commercial counterparts due to creative nature of software development).

[209]Bruce Perens, *The Open Source Definition*, *in* OPENSOURCES, *supra* note 2, at 182–83.

See also Lerner & Tirole, *Scope of Licensing*, *supra* note 98, at 2–3, 5 (showing relative restrictiveness of various licenses); *see supra* notes 142, 144 (elaborating on differences between Apache license and GPL).

licensed open-source software.[211] Taken as the GPL license intends, this colloquial use of "infect" means simply that the GPL license terms must be honored for all software in the modified work,[212] otherwise the license for the explicitly GPL-licensed open-source software is violated upon a redistribution of the software.

In other words, the GPL terms extend to other software combined with the GPL-licensed software. Rather than call this feature "viral," I call it the extension provision of the GPL. It exists due to the technological possibility to combine software. Software components can be mixed together in different forms to generate a computer program. The GPL wraps a legal issue around this technology capability. To better understand the legal issue requires a brief primer on the technology issue.

Recall that the source code is simply one form of the instructional composite. A computer program, that is, an instructional composite, directly or indirectly commands the computer to do something. In this characterization, the source code contains the indirect commands, while the object code, compiled from the source code, contains the direct commands. Each is a different form of the instructional composite. Open-source software can be combined with other software at either stage. Thus, a source code instructional composite could contain software under a variety of licenses, as well as public domain source code.[213]

For example, in the hypothetical GoneOutdoors open-source software, assume that Allen, Betty, and Carol are the only contributing developers, and

---

[211]The GPL does not use the word "viral" nor "infect," nor does it seek to apply its terms to other software that is merely aggregated with the GPL-licensed software. Its language expressing the "viral" extension provision is as follows:

> These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it. Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

GNU, GPL, *supra* note 99, § 2.

[212]*See* Margaret Jane Radin, *Humans, Computers, and Binding Commitment*, 75 IND. L.J. 1125, 1132 (2000) ("A viral contract (or attempted viral contract, because we do not know yet whether these attempts will result in an actual contract) is simply an attempt to make commitments run with a digital object.").

[213]Kem McClelland, *A Practical Guide to Using Open Source Software in a Time of Legal Uncertainty*, UNDERSTANDING ELECTRONIC CONTRACTING 2003 THE IMPACT OF REGULATION, NEW LAWS & NEW AGREEMENTS 351, 389–96 (Practicing Law Institute Intellectual Property Course Handbook Series, vol. 743, 2003).

that they have licensed GoneOutdoors under the GPL. Then Dan takes a copy of GoneOutdoors' source code and attaches to it his previously developed source code for Extreme Outdoor Sports. In effect, Dan mixes together his source code with Allen, Betty, and Carol's source code, which utilizes the GPL. Assume that the mixing causes minimal intermingling. As a result, if programmers wanted to, they could easily disentangle the two sets of source code. Next, Dan distributes the new program as open-source software, calling it GoneExtremeOutdoors. In the distribution, while Dan provides a complete set of object code, he only provides Allen, Betty, and Carol's source code. Dan is at risk of violating the GPL's extension provision. He has distributed his modified work "as a whole" but without providing source code for the whole.[214] Dan combined the two sets of software at the source code level. He had, however, an alternative design choice. He could have modified both sets of source code to interact, and then combined them after generating object code separately for each separate source code set. Doing this and then distributing GoneExtremeOutdoors with only Allen, Betty, and Carol's source code would also put Dan at risk of violating the extension provision, even though the software coupling was less intimate. The risk, however, might be lower because the coupling was less intimate. Thus, the extension provision seeks to apply the GPL license to other software combined with the software covered by the GPL, but only for certain means to combine, without precisely spelling out which means.[215]

There are other ways for software components to interact, coordinate, cooperate, and exchange data and signals as they execute in a computer. In effect, there is a continuum of possible coupling methods for software components. This technological fact raises the question: which of these coupling methods does the extension provision cover? Some of the software components might be covered by the GPL, others might not. If the non-GPL-licensed components are sufficiently coupled to the GPL-licensed software, then the GPL terms attempt to extend themselves to all coupled

---

[214]GNU, GPL, *supra* note 99, § 2.

[215]*See* Jianjun Deng et al., *Towards a Product Model of Open Source Software in a Commercial Environment*, 3rd Workshop on Open Source Software Engineering, International Conference on Software Engineering 31, 32 (Feller et al. eds., May 3–11, 2003), *available at* http://opensource.ucc.ie/icse2003/ (last visited June 19, 2003) ("[D]ifferent licenses impose constraints on the development, they even influence the architecture of a software that includes [open-source] parts as well as closed source parts.").

components.[216] This may or may not cause a violation of the GPL license terms. First, the non-GPL-licensed software may be licensed under another open-source license containing provisions that are compatible with the GPL. Compatible in this sense means that one who complies with the terms of this other license is also in compliance with the GPL terms. Second, the other license may directly forbid something the GPL provides, such as the right to redistribute. In this second example the GPL terms are violated. Third, the other software may not specify any license conditions, but its source code might not be made available. This would then violate the GPL, assuming that the extension provision applies the GPL terms to the other software.[217]

This example and the foregoing discussion demonstrate variances in prominent open-source licenses along four issues: source code, royalties, extension to other software, and mandatory reapplication of the terms to distributions. How these issues are implemented in the open-source license influences the collaborative possibilities for the software. These open-source license provisions, most notably the requirement that source code be made available, establish an ecology or culture for collaboration.[218]

Due to these licensing variances, to promote open-source software's growth, leaders in the open-source movement established the Open Source Definition ("OSD").[219] The OSD is a set of guidelines operated by a nonprofit

---

[216]One legal mechanism by which a coupled component potentially would need the permissions of the GPL is when the coupling establishes a derivative work under copyright law. Wacha, *supra* note 7, at 22–23. This is a vague and indeterminate boundary because defining what is and is not a derivative work for associated or intermingled software components is difficult. If the coupling establishes a derivative work, then the GPL would have the legal power to require that the coupled component also be licensed under the GPL's terms. If the coupling does not establish a derivative work, the situation is less clear and would require additional analysis based on other factors.

[217]In practice, for the GPL, industry custom helps define the boundary for the extension provision. *See* Torvalds, *supra* note 116, at 108–09 (noting that what counts as derived work under GPL can be vague, Torvalds describes that "[w]e 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."); Wacha, *supra* note 7, at 22–23 ("The area open to the broadest interpretation, and the most intense debate, surrounds when and how proprietary code can coexist with GPL code.").

[218]Kogut, *supra* note 178, at 249–50.

[219]*Id.* at 255.

entity as a certification program.[220] While not a license, the OSD guidelines take positions on the four issues and thus add a third perspective. To illustrate, the table below aligns the three approaches, Apache,[221] OSD, and GPL, for the four issues.

[220]*See supra* note 168 (discussing OSD as guidelines). The definition itself specifies ten conditions: (1) free redistribution, which must be provided; (2) source code, which must be available; (3) derived works, which must be allowed; (4) code integrity, relating to a technical point about attribution for "patched" source code; (5) no discrimination against persons or groups; (6) no discrimination against fields of endeavor; (7) distribution of license, meaning that the conditioned permission to use under the license cannot be further conditioned by redistributors; (8) license must not be specific to a product; (9) the license must not restrict other software, meaning that it cannot insist that all software merely distributed with it be open-source; and (10) the license must be technology-neutral, meaning that the license cannot depend on technology-enabled assent mechanisms such as those found in "click-wrap" agreements. The Open Source Definition, ver. 1.9 (2003), *at* http://opensource.org/docs/def_print.php (last visited June 30, 2003) [hereinafter OSD 1.9]. *See also* Bruce Perens, *The Open Source Definition*, *in* OPENSOURCES, *supra* note 2, at 176–80 (analyzing OSD provisions and providing further commentary on each). OSD 1.9 notes in an annotation to section nine that the GPL meets this requirement. *Id.* at § 9. Indeed, the GPL itself acknowledges that mere distribution with other software does not rise to the level of coupling that would trigger the GPL's extension provision. GNU, GPL, *supra* note 99, § 2 ("[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.").

[221]To give full attribution for the Apache license I acknowledge its lineage. The license derives from the BSD license. Researchers who developed a free version of the Unix operating system used the BSD license to distribute the free Berkley Unix. *See supra* note 144 (stating that Apache license is derived from BSD license for the BSD flavor of UNIX). The Apache project, starting with research-institution code, continued this approach. Bruce Perens, *The Open Source Definition*, *in* OPENSOURCES, *supra* note 2, at 183.

Issue	Apache	OSD	GPL[222]
source with redistribution	not required	required	required
royalties	not prohibited	prohibited	prohibited
extension provision	implicitly required, effect is minor	no	yes
reapplication of terms	implicitly required, effect is minor	must be allowed, not required	required

Table 1 Key Open-Source Software License Variances

On a continuum, the Apache license is the least restrictive. The GPL has the most "restrictions," but they have the effect of insulating the software from privatization and bringing maximum pressure against withholding source code.

[222]The Free Software Foundation ("FSF"), as part of its GNU operating system project, promulgated the GPL, but it also promulgated a less "restrictive" license that sits to the left of the GPL on the continuum: the GNU Lesser General Public License. Free Software Foundation, GNU Project, GNU Lesser General Public License, *at* http://www.gnu.org/copyleft/lesser.html (last visited June 4, 2003) [hereinafter LGPL]. The introductory remarks to the LGPL note that it has been renamed. Version 2.0 of the license had the name "Library General Public License," but version 2.1 has the current name. The earlier name denotes the license's original earmarking for software libraries—components specifically designed to be combined with other software. The remarks also discuss the primary difference between the GPL and the LPGL: the latter lacks a strong extension provision.

> This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs. 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.

LGPL. Thus, the LGPL presumes that certain types of software component coupling, namely linking with a library, creates a derived work to which license terms would extend. The FSF thus provides an alternative license, the LGPL, that does not extend the GPL provisions of no royalties and source code availability to the coupled "whole" although source code for the software library itself must be made available. The FSF, despite promulgating the LGPL, has advocated against its use because it believes that that LGPL "does Less to protect the user's freedom than the ordinary [GPL]" and finds use of the LGPL justified in only a few special cases. *Id.*

The OSD is in the middle.[223] The Apache license is only one page whereas the GPL is seven pages. The effect of the Apache license's implicit extension provision and reapplication provision is minor because the license only imposes obligations of attribution and notice. Thus, while these obligations might extend to a much larger work in which the software is incorporated, the cost of compliance is de minimis. There is no opportunity cost as with the GPL's extension provision, where redistributors may lose the chance to privatize or keep private (charge royalties and keep the source code secret) other software if they couple the modified or unmodified GPL-licensed software with other software. Losing the privatization opportunity occurs through the interaction of the extension provision and the reapplication provision. The GPL says that, because of the extension provision, the software with which one has coupled the original GPL-licensed software must now be under the GPL.[224] Once under the GPL, future redistributions must also be under the GPL.

The OSD guidelines do not require the extension provision and take a flexible approach to the reapplication of terms provision. An open-source license is certified as compliant with the definition if the license at least allows future redistributors to reapply the same terms. They need not do so, but if they do, they are also compliant. Under this logic, the GPL complies with the OSD. Thus, the OSD acts as a baseline. A license may require more, as the GPL does, to promote source code disclosure, but it need not do so to be certified.[225]

By leaving out dozens of licenses, this discussion understates the degree of license variance, but it does capture the extremes: the minimal Apache license and the expansive GPL.[226] Given this variety, a natural question is the impact of license variance on collaboration, especially given that the Apache project has successfully managed collaborative activity with a minimal license just as the Linux development, under the GPL, has successfully collaborated.[227] The attributes of the collaborative culture will vary due to many factors, but license differences are reasonably thought to be an influencing factor. Thus,

---

[223]The OSD is effectively silent concerning an extension provision. In section nine, entitled, "License Must Not Restrict Other Software," the OSD is clear that the license cannot impose restrictions on co-distributed software. OSD 1.9, *supra* note 220, § 9. The OSD also effectively states that the GPL's extension provision does not render the GPL noncompliant with section nine of the OSD. *Id.*

[224]Wacha, *supra* note 7, at 22–23 (discussing various technical methods of associating software that may represent legally significant degrees of coupling).

[225]*See supra* note 223 (describing OSD extension provision).

[226]*See* Robert W. Gomulkiewicz, *De-Bugging Open Source Licenses*, 64 U. PITT. L. REV. 75, 82–83, 92–93 (2002) (noting that programmers often choose either GPL or Apache-style license/BSD-style license, and discussing differences between these two).

[227]Kogut, *supra* note 178, at 257 (arguing that under GPL, used by Linux, any balkanization of code does not become proprietary, whereas, Apache, although it has lesser strength license, has stronger governance structure and thus has avoided balkanization via its governance mechanisms).

the next Subsection reviews the effect these licensing differences may have on collaboration.

(b) Collaborative Implications of Licensing Differences

Among the four licensing issues discussed, source code availability is the most important collaboration enabler, but it also creates a risk for a feared anti-collaborative event: forking. An open-source project forks when a group takes the product in a new direction, establishing its own separate source code base, and developing the new product apart from its parent.[228] Forking is a possibility as long as the source code is available, which it is under almost all open-source software licenses.[229]

The forking possibility is inherent in the open-source approach. For example, in the hypothetical GoneOutdoors open-source software, assume that Allen, Betty, and Carol are the only contributing developers, and that they have licensed GoneOutdoors under the GPL. Then Ed, Fran, and Gill take a copy of GoneOutdoors' source code and significantly change and alter the code. But for some reason they do not want to work with Allen, Betty, and Carol, and thus do not contribute their changes to GoneOutdoors. Instead, Ed, Fran, and Gill make their software available from a different Web site, calling it WentNatural.[230] This is completely legitimate under the GPL, although it may not be optimal or efficient because the aggregate effort for the software is now

[228]McGowan, *supra* note 4, at 263, 278 (analogizing forking as one way for one to "take ownership" of project, and noting that "[n]orms against forking reward conduct that makes production of code smoother and collectively more efficient and penalize conduct for which individual returns are likely (on average) to exceed the gains to the code base. Such norms tend to keep the community's focus on code rather than individual income."); *see* Raymond, *supra* note 125, §§ 8, 13 (discussing motivations for individual developers and entities involved with open source).

[229]The Apache license is an exception: it does not require that redistributors make the source code available. Apache License, *supra* note 141.

[230]If Ed, Fran, and Gill called their software GoneOutdoors, this creates a more than de minimis risk of trademark law issues between the two groups.

split.[231] Some in the open-source community see forking as a threat mechanism that helps discipline a project's leaders, but actual forking is rare.[232] So, while source code availability and the open-source approach make anti-collaborative forking a possibility, other factors seem to minimize forking frequency.

For example, open-source licenses that prohibit royalties for the software may blunt the incentive to fork a project and thus have a pro-collaborative effect. The OSD explicitly cites this incentive as a reason to require that its certified licenses prohibit royalties.[233] The experience of Unix is evidence for this point. One of the major flavors of the Unix operating system sourced from the University of California at Berkley utilized a license similar to the Apache license.[234] In fact, the Apache license is styled from the license used for the Berkley "flavor" of Unix.[235] As a result of the minimally restrictive Berkley Unix license, many private entities developed their own sub-flavor of the Berkley Unix, resulting in further fragmentation of the code base, the operating system, and the brand identity, and inhibiting, as one would expect, inter-entity collaboration.

In addition to a possible anti-forking effect, an anti-royalty open-source license provision is also thought to promote contributions to the open-source software because contributors can be confident that others will not reap what they have sown, in the sense of extracting private rents for the contributed work.[236] This is a necessary precondition for the large scale collaboration exhibited by projects such as Linux, and is in essence a codified norm of the

---

[231]Whether the hypothetical fork of GoneOutdoors to WentNatural is in fact non-optimal or inefficient depends on a number of factors. One is whether there are sufficient economies of scope to warrant combining the functionality of the two software projects. The more that WentNatural is internally and externally differentiated from GoneOutdoors, the greater likelihood that it is in fact efficient for it to be a separate project. Another factor is the synergy within each development group, compared to the synergy that would exist if the two groups worked together. Additionally, ideological factors could also play a role and cause groups to believe that they could not work together. A variety of other factors could pertain to the issue, each expressing the general theme that the payoff from the collaboration under each alternative, forking or no forking, would influence in fact whether forking occurred. While I have characterized forking as anti-collaborative, in some of the instances just described forking may in fact be pro-collaborative because it "right-sizes" through self-selection the group that will focus on its self-defined open-source software project. *See generally* Myatt & Wallace, *supra* note 89, at 448–49 (modeling collective action problems among developers of open-source software project and noting interrelationship among project's size and its level of integration, i.e., whether it is administered in components or as whole, and programmer coordination).

[232]*See* Raymond, *supra* note 125, § 8 (noting that "forking is frowned upon and considered a last resort").

[233]OSD 1.9, *supra* note 220, § 1 ("By constraining the license to require free redistribution, we eliminate the temptation to throw away many long-term gains in order to make a few short-term sales dollars. If we didn't do this, there would be lots of pressure for cooperators to defect.").

[234]*See supra* notes 117–118, 144 (giving history of UNIX, BSD, Apache, and GPL).

[235]Bruce Perens, *The Open Source Definition*, *in* OPENSOURCES, *supra* note 2, at 183.

[236]*See* Bessen, *supra* note 5, at 13 (noting developer's desire to help community).

open-source community, both for individuals in the community and for entities that operate there.

The other two license issues, the reapplication provision and the extension provision, have a less clear collaborative impact. The reapplication provision is probably collaboratively neutral in the sense that if the license terms are generally pro-collaborative, then mandatory reapplication should extend the pro-collaborative effect. If the license terms discourage collaboration, then that effect also probably carries forward. The extension provision certainly impedes use of open-source code in proprietary software. Most private software entities would be unwilling to place an entire program under the GPL merely to obtain the benefit of using a small amount of GPL-licensed code in the program, regardless of how efficient or well designed one found the GPL-licensed code.[237] Whether impeding this use of GPL-licensed software is anti-collaborative or pro-collaborative is probably a matter of perspective. Projects like Apache or the Berkley Unix provide what is almost a public domain input for the production process of those who use these projects' code in private software. In one sense, this enables collaboration, but the collaboration runs only in one direction: the open-source software helps the private entity programmers learn and apply new and better software. The private entity programmer does not, and is not allowed to, contribute back to the open-source project.

Many factors beyond these four license issues will influence the degree and success of collaboration because it is a multi-causal phenomenon. A few of these are of note because they are linked to the license issues. First is the incentive of ongoing users and sometimes contributors to eliminate the cost of keeping keep parallel revisions for their custom modifications when they also want to partake in future functionality from the project's later versions.[238] Most open-source licenses allow such users to keep their modification private to

---

[237]FLOSS Activities, *supra* note 152, at 28. ("As firms can protect most of their intellectual property in the domain of software development, it can be assumed that they only give as much intellectual property away in the form of Open Source software as is optimal for them.") This study goes on to note:

> One issue pointed out, for example by Microsoft, is the viral nature of the GPL (which governs Linux) and especially ambiguities in its vitality, which supposedly makes it difficult to build commercial software on top of Open Source software. While a discussion of this legal issue is beyond the scope of this paper, it has to be taken in account that unclear legal implications might indeed be issues keeping companies from taking part in those Open Source projects governed by such licenses or from including such software as infrastructure components into their products.

Id. (citation omitted).

[238]*See supra* note 138 and accompanying text (discussing motivation for submitting changes and avoiding back-fitting costs for new versions).

their own use,[239] but there is an incentive to contribute the modifications to the project if the cost of maintaining them separately is greater than any competitive advantage derived from holding them private and, if contributed, they will be incorporated into the project by its leaders. Second is the practical necessity for a project initiator or initiating group to develop a critical mass of code to attract contributors.[240] Sufficient framework and vision for the project must exist to enable programmers to evaluate the opportunity and determine how they could apply themselves, to their satisfaction, to the project. Third is to ensure that even though source code is technically available that it is not obfuscated or hard to obtain.[241] For example, open-source licenses that require attribution for modifications promote commenting or other mechanisms to record the lineage of the source code. In addition, some licenses specify what they mean by source code in order to ensure that items necessary to compiling and assembling the software are made available along with the computer program instructions themselves.[242]

The open-source approach provides the foundation for collaboration, but does not provide the method and means for collaboration. These are both new and old in open-source software development. Some traditional software development techniques live on in new forms, such as some degree of organizing hierarchy, but it is a distributed and informal hierarchy foreign to traditional software development.[243] Truly new is that the source code must be available, and that royalties are prohibited. These conditions, combined with

---

[239]*See* Miller, *supra* note 7, at 497–98 (noting that GPL allows modifications to be kept private unless redistributed).

[240]George N. Dafermos, *Management and Virtual Decentralised Networks: The Linux Project*, 6 FIRST MONDAY No. 11, § 6 (Nov. 2001), *available at* http://www.firstmonday.dk/issues/issue6_11/dafermos/ (last visited June 14, 2003) (describing four conditions needed to mobilize critical mass of resources).

[241]The Apache license does not require that redistributors make source code available; but, through its mandatory attribution provisions, it suggests to users where they might find the original source code from which the programmer built the software. *See* Apache License, *supra* note 141, § 2 ("Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.").

[242]*See, e.g.*, GNU, GPL, *supra* note 99, § 3 ("For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable.").

[243]There are other ways to express the contrast between traditional software development and open-source development. One is the observation Mike Madison made to me that traditional copyright assumes the traditional development method, while open-source licensing, or "copyleft," can dispense with the hierarchy necessary for software development. Open-source can achieve the same (or better) output with a different organizational structure. *See* Benkler, *supra* note 4, at 378–80 (discussing automated integration and iterative peer productions of integration as mechanisms that succeed and sustain themselves). If we can dispense with the traditional hierarchy, then we can and should evaluate whether the legal rights supporting open-source software development should spring from a new basis.

the rise of the Internet, have enabled collaboration to a degree that has shocked the information technology world, exemplified by the success of Linux and Apache. However, the foundation, the open-source approach, for all its beneficial impact on collaboration, is a mix of the old—copyright law and licensing law—applied in new ways. The final Subsection of this Part will briefly examine some of the other legal implications that this new mix raises.

5. Other Legal Considerations for Open-Source Software Licenses

The copyright-based open-source approach communicates a conditional permission to use the software through an open-source license, which raises a number of doctrinal questions. In this Subsection I will highlight some of the issues and their analysis by commentators to demonstrate the degree and range of questions posed by the nascent open-source approach.

One question is whether the license is also a contract, and if so, whether, or to what extent it is enforceable.[244] The enforceability question should probably be analyzed on a term-by-term basis for the license under examination. Commentators regularly state that the most prominent open-source license, the GPL, has not been tested in court.[245] The GPL itself notes that it is not normally implemented as an agreement that obtains assent from the user.[246] The doctrinal contract questions here are similar to enforceability

[244]*See generally* McGowan, *supra* note 4, at 289–302 (examining open-source approach and contract law implications).

[245]Heffan, *supra* note 5, at 1509; Lee, *supra* note 131, at 57; McGowan, *supra* note 4, at 243; Wacha, *supra* note 7, at 23. While no court cases have yet interpreted the GPL, it has been at issue in several disputes in which there was some action in the courts, but without generating an opinion discussing the issues. *See* Progress Software Corp. v. MYSQL AB, 195 F. Supp. 2d. 328, 329–30 (D. Mass. 2002) (denying injunctive relief on copyright counterclaim because withheld source code was subsequently disclosed); *see also* Michael J. Madison, *Reconstructing the Software License*, 35 LOY. U. CHI. L.J. 275, 294 (noting open-source model's lack of testing in courts); McGowan, *supra* note 4, at 300–01 (discussing case involving assignment of code released under GPL).

[246]GNU, GPL, *supra* note 99, § 5 ("You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works.").

questions for shrinkwrap licenses.[247] David McGowan notes that the GPL "seeks to create binding obligations on downstream code through notice-and-use provisions," and goes on to analyze several questions this regime raises from a contract law perspective, while noting that, even if the contract analysis leaves gaps, the copyright protections still remain.[248] McGowan's emphasis, however, is not to answer these questions, but to raise them, while focusing elsewhere: "The main point of the GPL is the social structure it supports—the opportunities it creates, the practices it enables, and the practices it forbids."[249]

The open-source approach raises several doctrinal twists that are new even compared to shrinkwrap license analysis. McGowan analyzes issues of formation and assent, whether downstream users are bound or whether issues of privity hinder this, and questions of term, termination, and assignment.[250] Two themes pervade this analysis. First, the possibility that, uniquely for open-source software, a project might generate a chain of takers or licensees.[251] Everyone in the chain is at risk of a copyright infringement suit by the original programmers and any contributors along the way. In that regard the contract

---

[247]Bobko, *supra* note 13, at 100–03; Lee, *supra* note 131, at 72–79; McGowan, *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 private software or Web sites to present users with an opportunity to affirm terms of the license. *See* OSD 1.9, *supra* note 220, § 10 (requiring that "[n]o provision of the license may be predicated on any individual technology or style of interface" because section ten is "specifically aimed 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 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, *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, 1034–48, 1054–76 (1998).

[248]McGowan, *supra* note 4, at 289 ("If the GPL is ineffective, the copyright still persists."). *See also* David McGowan, *Legal Aspects of Free and Open Source Software*, 10–14 (Feb. 2004), *at* http://www.law.umn.edu/uploads/253/McGowanD-OpenSource.rtf (discussing possibility of contract formation and other issues, such as term of agreement, using GPL as paradigmatic example of open-source software license).

[249]McGowan, *supra* note 4, at 302.

[250]*Id.* at 289–302.

[251]The chain is only one possible distribution pathway, and perhaps not the most common. The more common pathway for active projects with a user base is a web of distribution, with some modifications coming back to a centralized repository to be incorporated into the "official" distribution of the project. Recall the GoneOutdoors software project of Allen, Betty, and Carol, where Carol posted the code on a Web site and the group of developers swelled as others joined the project. *See supra* Figure 2 and accompanying text (showing interdependence of individuals of open-source software). As each contributor took a copy of new versions, they used a work, some of which was their copyrighted material. The rest was copyrighted to the other contributors. In this configuration, all were now interdependent on the open-source approach.

analysis is not the primary concern. However, the doctrinal contract analysis demonstrates that contract law may bear on the situation,[252] or at least demonstrates that traditional contract recourse may not be available to a licensee or taker in this chain.[253] Second, the analysis shows that the issues raised by the open-source approach establish just enough novelty to create uncertainty. For example, McGowan's discussion of assignments suggests one way to manage the uncertainty that results from the distributed ownership produced by the open-source approach:

> One way to combat opportunism is to ask authors of open-source code to assign their rights to an organization controlled by a representative portion of the community. The organization would then decide whether to terminate rights or take code private and, if

---

[252]As McGowan discusses, one question is whether a user, who is not a contributing programmer, would have contract recourse if the programming group, who held all copyright interests in the open-source software, were able to organize an effort to take the code private. Would this group be able to "revoke" the license to users? If so, would such users have recourse under contract law? *See generally* McGowan, *supra* note 4, at 289–97 (analyzing these questions from perspective of case law bearing on shrinkwrap license agreements).

[253]For open-source software users or marginal contributors to a project, the possibility of no contractual recourse if the project leaders decided to privatize the project is likely a risk that most are willing to bear, because the norms of the open-source community are diametrically opposed to such behavior. Moreover, even if there is not a strict contract claim, the law of remedies might provide recourse.

> [S]uppose that an author distributed code under the GPL and members of the open-source community worked to improve the code by fixing flaws and writing additional code. Suppose further that, at some point after considerable improvements have been made, the author claims that the GPL is ineffective to grant enforceable licenses to create derivative works. If the author attempted to take the improved version of the code private, equitable theories such as estoppel might provide a useful backstop in cases where the facts could not support a formal contract theory.

McGowan, *supra* note 4, at 297.

McGowan also notes that because the agreement term is not specified by the GPL, some states' contract law might make the license terminable at will. *Id.* at 298–99. As a result, "the potential ability to terminate at will increases the risk of opportunistic behavior by rights holders." *Id.* at 299. Licensees might be better off if no contract is formed so that "community members could rest on estoppel arguments," but those who created derivative works before the attempted license termination should have their rights preserved. *Id.*

> properly constituted, its decisions might reasonably reflect community sentiment.[254]

If an open-source programmer were to assign her copyright ownership to a supposedly pro-open-source organization and the organization took the open-source project and made it private, the programmer's remedy would then squarely fall within contract law, assuming that the organization promised to safeguard the software according to the open-source approach.[255] Thus, an assignment of the contributor's copyright raises a number of intriguing questions, and McGowan identifies other ways that assignments could complicate, or be complicated by, the open-source approach.[256]

Others areas of concern and uncertainty include warranties[257] and whether copyright law preempts the open-source license (when viewed as a contract).[258]

---

[254]*Id.* at 300. McGowan further notes that the Free Software Foundation advocates the assignment approach. *Id.* In discussing the implications of a "complete absence of property in the software domain" for open-source software, Benkler notes that "copyright permits free software projects to use licensing to defend themselves from defection." Benkler, *Coase's Penguin*, *supra* note 4, at 446. He then makes a point similar to McGowan's about the possible value of public mechanisms for preserving open-source software: "The same protection from defection might be provided by other means as well, such as creating simple public mechanisms for contributing one's work in a way that makes it unsusceptible to downstream appropriation—a conservancy of sorts." *Id.*

[255]In this scenario, although the primary remedy would be under the programmer's assignment contract with the organization (assuming the contract contained promises to observe the open-source approach), depending on the situation, the programmer might also have recourse under beneficial ownership in relation to standing doctrines under copyright law. *See* NIMMER & NIMMER, *supra* note 55, § 12.04[B]–[C] (discussing copyright infringement standing doctrine and brining suit as beneficial owner).

[256]*See* McGowan, *supra* note 4, at 301–02 & n.283 (discussing reverse-engineered code that partially disabled filtering software, which was released under GPL license, and then assigned to plaintiff (and owner of filtering software) as the result of copyright infringement suit against two original programmers who developed code and released it).

[257]For example, the GPL requires that warranties be disclaimed.

> [F]or 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.

GNU, GPL, *supra* note 99, § 1. One commentator, however, has questioned the effectiveness of this scheme. *See* Gomulkiewicz, *supra* note 226, at 86 ("If the GPL is not the contract that governs a user's right to run the software, Uniform Commercial Code Article 2's implied warranties may apply to the transaction and consequential damages would be available as a default rule.").

Thus, an interesting set of issues potentially lurk behind open-source software licenses such as the GPL. However, equally interesting is the vacuum of court cases on these subjects. The vacuum is reported to exist in part due to the success of the Free Software Foundation in mediating disputes.[259] This success is certainly beneficial for the open-source movement, even if it leaves some curious as to how a court might resolve the issues highlighted above.

The identification of these potentially uncertain issues raises the question: what is their impact on open-source collaboration? Since there is no empirical information to apply to the question, one can only predict that to the extent the uncertainty is perceived, it might make programmers less likely to contribute if they fear some partial breakdown in the licensing scheme upon which their efforts are insulated from private appropriation. Certainly some programmers are aware, to some degree, of the various licenses—they must choose a license when they initiate a project.[260] Thus, there is some recognition that different

---

Even disclaiming warranties and damages, however, does not fully protect open-source programmers from claims of implied warranty. The risk is not uniform, since it goes by state law. Thus, relevant to the issue are revisions to state law as a result of the National Conference of Commissioners on Uniform State Laws ("NCCUSL") project which has generated the Uniform Computer Information Transactions Act ("UCITA"). *See* NCCUSL Web, *at* http://www.nccusl.org/nccusl/ (last visited July 6, 2003) (describing NCCUSL and providing links to UCITA project). Illustrating the implied warranty issue is Maryland's adoption of UCITA but with revisions to account for open-source software: "[n]o implied warranty of merchantability is given where a product is distributed for free unless the product is distributed in conjunction with some other sale or lease." Charles Shafer, *Scope of UCITA: Who and What are Affected?*, UNIFORM COMPUTER INFORMATION TRANSACTIONS ACT: A BROAD PERSPECTIVE 325, 248 (Practicing Law Institute Intellectual Property Course Handbook Series, vol. 672, 2001). Later, the NCCUSL UCITA committee recommended "a new section that exempts from implied warranty rules the transfer of a computer program where no contract fee is charged for the right to use, copy, modify or distribute the program." Report of UCITA Standby Committee, § 3(F) (2001) (Recommendation 10), *available at* http://www.nccusl.org/nccusl/UCITA-2001-comm-fin.htm (last visited Feb. 4, 2002). To the extent the open-source license is enforced under a contract model, one commentator has noted that another aspect of UCITA may bear on open-source software, suggesting that a UCITA state would provide heightened background law support for the license. ROSENBERG, *supra* note 35, at 239 ("Open Source fans can be happy that their licenses will now be taken more seriously . . . .").

[258]Bobko, *supra* note 13, at 103–05 (arguing that GPL is not preempted by copyright law).

[259]Webbink, *supra* note 8, at 683. *See also* Moglen, *Enforcing the GPL I & II*, *supra* note 167 (discussing Professor Moglen's efforts as counsel to Free Software Foundation to enforce GPL).

[260]*See generally* Lerner & Tirole, *Scope of Licensing*, *supra* note 98, at 2–4, 8–20 (exploring "the various considerations that figure into the licensor's decision of how restrictive a license to employ" and noting that ambiguities remain about open-source licenses).

licenses will have different effects, and perhaps even recognition at some level that different licenses may have differing degrees of efficacy.[261]

Despite the questions, successful—that is, widely used—licenses, such as the GPL, cast a long and important shadow, particularly when they self-perpetuate by including a reapplication provision.[262] They establish a foundation anchoring the open-source community's practices. They enable and facilitate distributed software development by ensuring that source code, the most important form of the computer program's instructional composite, is available. This anchoring embodies norms of the open-source community—a collection of individuals and groups who have generated both a new software development approach and a new ideology to support the approach. The ideology allows, and in some corners embraces, commercial activity as long as the software remains open and free for all to use. Commercial entities have responded to fill the aggregator and distributor role for the most popular and foundational open-source software, such as Linux. The grand result is an expanding system of software production and distribution that provides a viable model to compete with traditional software development. But it is a result that depends on, first and foremost, access and availability to the source code to engender the collaborative effort.

The emphasis on source code leads to the next step in the comparison. Just as the open-source approach requires the work to be viewable in a particular way, the copyright tradition in civil law jurisdictions provides rights that allow for control over the view that a work presents. These rights are historically strange to our system of copyright. Even as international harmonization pushes copyright to a common denominator, these rights are minimally implemented in our system. Thus, the next Part reviews moral rights of authors and artists in the civil law tradition.

IV. AUTHORS' AND ARTISTS' MORAL RIGHTS

Having examined the significance of source code and how it underpins open-source software, this Part turns its attention to my point of comparison:

---

[261]Since commentary on the GPL has been extensive, and, if legal commentary, often mentions that the GPL has not been "tested" in court, some project initiators might shy away from such a license with "bad" publicity. Additionally, some guidance has been published on choice of a license, including the OSD certification efforts, which would counsel one to choose licenses that meet the definition. *See supra* notes 168 and 220 (explaining OSD as guidelines to determine whether license meets OSD).

[262]*See* Ryan, *supra* note 170, at 683 (noting that "shrinkwrap" licenses, and other mechanisms of control over copyrighted works, transform law governing access to information in copyrighted works from public law copyright regime to private law regime).

the traditional civil law *droit moral*, or moral right of attribution[263] and integrity. These two rights exist among other rights in the *droit moral* set, but they are the focus because they parallel the open-source approach. They give authors control over aspects of their work in ways analogous to open-source software's governance of source code. The legal scholarship on moral rights is extensive; I do not review it in full, but rather sample it to set a base for my comparison.

I proceed in this Part as follows: the first Section generally introduces moral rights with emphasis on the right of attribution and the right of integrity. Among the four prominent moral rights, in my comparison, these two are the most visible in the open-source approach.[264] I present a brief review of their emergence and place within the generalized copyright regime of civil law systems. Next is a more detailed review of the right of integrity, followed by a discussion of moral rights in software. Unlike United States copyright law, some civil law jurisdictions extend limited moral rights protection to software. The United States is resistant to moral rights generally,[265] which translates into a lack of explicit moral rights for software and other literary works. Other jurisdictions, having moral rights as a core part of their tradition, extend these rights to software to varying degrees.[266] This is just one distinction among many demonstrating the ideological and doctrinal differences of moral rights compared to our copyright tradition.

A. Moral Rights in the Civil Law Tradition

There are four moral rights in the civil law tradition: (1) the right to publish the work (that is, to determine when it is first divulged); (2) the right to have the author's name, and no other name, attributed to the work; (3) the right to object to impaired integrity of the work (that is, mutilations, modifications, or distortions of the work detrimental to the author's or artist's honor or

---

[263]The right of attribution sometimes is referred to as the right of paternity. PHILLIPS ET AL., *supra* note 16, at 59.

[264]While the rights of attribution and integrity are the most visible, the right to publish has analogs in the open-source approach: one need not distribute one's modifications to open-source software to anyone if one does not wish to do so. Users who modify the software can simply use their changes in isolation. On the other hand, the moral right to sometimes withdraw work on equitable terms has no analog in the open-source approach. In addition, there is another right sometimes grouped with the set of moral rights: the *droit de suite*, or royalty right, which allows "an author who has sold a painting, sculpture or other object embodying his work to receive a proportion of the proceeds of any subsequent resale of that item." PHILLIPS ET AL., *supra* note 16, at 56. The royalty right has an exact opposite implementation in the open-source approach—the open-source license disallows royalties on use of the software.

[265]Françon, *supra* note 27, at 75.

[266]Arthur Fakes, *The EEC's Directive on Software Protection and its Moral Rights Loophole*, 5 SOFTWARE L.J. 531, 546 n.23, 554 n.42, 609–612 (1992).

reputation); and, most obscurely, (4) the right to withdraw the work in certain situations on equitable terms.[267]

These rights to some degree exist apart from the author's or artist's economic copyright rights in the work.[268] Thus, a work's creator could sell the physical embodiment, transfer away the economic rights in the work, yet retain moral rights.[269] Replaying my example in the Introduction, if a sculptor in France creates a statue, the sculptor could sell it, then assign away the economic copyright rights, yet retain moral rights, such as the right of integrity. Then, if the buyer mutilated the statue by painting it purple, the sculptor may have legal recourse to remedy the mutilation caused by the purpling.[270] Thus, the right of integrity allows the sculptor to govern the view that the work presents.

This small example often amazes those first learning about moral rights—especially if they are steeped in the traditions of common law alienability of property and preferences for transferability of rights.[271] The next Subsection will describe how in civil law systems this species of protection for creative and expressive works emerged in separate coexistence with economic rights in such works.

---

[267]Hansmann & Santilli, *supra* note 17, at 95–96; Treece, *supra* note 26, at 487–88, 494, 499–500. Artists' and authors' moral rights vary by jurisdiction, thus I resort to the Berne Convention's provisions when a common expression of the right of attribution and the right of expression is necessary. *See* Berne Convention, *supra* note 28, § 6bis(1) (describing treaty's expression of right of attribution and right of integrity). Françon describes the French implementation of these four rights, calling what I term the right of attribution the right of authorship, and calling the right of right of integrity the right to respect. *See* Françon, *supra* note 27, at 75. The French right to respect is broader than the Berne Convention right of integrity. Dietz, *supra* note 28, at 200–01, 203 (calling Convention's approach "minimalist" compared to French approach); Sheldon W. Halpern, *Of Moral Right and Moral Righteousness*, 1 MARQ. INTELL, PROP. L. REV. 65, 71 (1997) (calling French protection "broader").

[268]The separation of moral rights from copyright rights is exemplified by France and Germany. Germany's moral rights are thought to follow a unitary or monist theory—it is minimally "possible to distinguish the limits of each element" and under a unitary approach moral rights and copyright are seen "as two facets of a single right." PHILLIPS ET AL., *supra* note 16, at 16. France, on the other hand, fits the dualist theory, which "recognizes in the author's right the elements of two different orders. There is a separation of the author's right to assert his creative relationship to his work and his right to put the work to economic use." *Id. Accord* GOLDSTEIN, *supra* note 20, at ix.

[269]Ciolino, *supra* note 26, at 937 ("Moral rights permit the author of a work to protect it even after the work has been sold to another.").

[270]*See, e.g.*, Halpern, *supra* note 267, at 71 (noting similar example).

[271]Ciolino, *supra* note 26, at 937; Netanel, *supra* note 18, at 356–58 (describing Anglo-American traditions of property alienability). Moral rights permit "the artist, in effect, to maintain a continuing negative servitude in his work, analogous to the servitudes that can be created in real property in both civil-law and common-law systems." Hansmann & Santilli, *supra* note 17, at 101.

1. European Development

To put things in the language of open-source software, and somewhat oversimplifying, copyright in continental Europe forked beginning around the time of the French revolution, resulting in a two-pronged system on the Continent, but not in England.[272] One prong envisioned that copyright protection was pecuniary, under the argument that some exclusive rights must attach to copyrighted works to create an incentive for their production.[273] As this fork developed, so did a second basis for assigning authors' rights to control their works. This second basis underpinned what today manifests itself as moral rights.[274] While there is disagreement as to whether the fork was partial or full, to what it owes its ideological basis, and whether it is truly separate from the pecuniary copyright interests,[275] it is clear that modern civil

---

[272]*See., e.g.*, Ciolino, *supra* note 26, at 938–39 ("Although *droit moral* developed in civil-law jurisdictions, it is a judicially-created doctrine that developed in the wake of the French Revolution to assure that artists' rights no longer existed at the pleasure of the sovereign.") (citations omitted); Ginsburg, *supra* note 36, at 991–92, 996 (comparing civil and common law copyright schemes); Hugh C. Hansen, *International Copyright: An Unorthodox Analysis*, 29 VAND. J. TRANSNAT'L L. 579, 580 (1996) ("Overall there were two systems: (1) the Anglo-American so-called 'economic' system and (2) the French and Continental 'author's rights' system with its concomitant fascination with 'moral rights.'"); Peeler, *supra* note 29, at 436–41, 447–48 (describing succession of French cases where court's attitude toward literary property evolved from lower tier or property, chartered by state to encourage production of works, to higher form or property inherent in creative effort). England did not statutorily implement author's moral rights until 1988; it did so in order to join the Berne Convention. PHILLIPS ET AL., *supra* note 16, at 15, 17–18, 58. Peeler describes the schism that developed in French law which gave rise to moral rights: "[T]hese rights of personality in works of art were not part of France's original copyright scheme, nor were they explicitly part of the pre-statute philosophical debate. Instead, the source of these rights is the nineteenth century French court decisions." Peeler, *supra* note 29, at 454.

[273]I greatly oversimplify in sketching the development of a schism in continental copyright resulting in two types of rights—copyright and moral rights. The degree of schism in each country varies. *See* GOLDSTEIN, *supra* note 20, at 8–9, 283–85 (comparing various countries' recognition and protection for moral rights). Moreover, two theories, monist versus dualist, competed for the underlying justifications for moral rights, resulting in consequences for statutory implementation because France adopted a dualist approach whereas Germany's approach was monist. Ciolino, *supra* note 26, at 939–40; PHILLIPS ET AL., *supra* note 16, at 16; *see also supra* note 268 (describing separation of morals rights from copyright rights). Thus, the moral rights jurisdictions were not themselves uniform, in addition to differing with non-moral-rights common law jurisdictions.

[274]*See* GOLDSTEIN, *supra* note 20, at 3–4 (describing conventional wisdom that copyright and author's rights (moral rights) spring from different justifications: utilitarian for copyright versus "a matter of right and justice" based on natural rights philosophy for moral rights).

[275]*See, e.g.*, GOLDSTEIN, *supra* note 20, at viii–ix, 4 (challenging conventional wisdom of divided philosophies for copyright and moral rights: "it is not clear that the division was ever more than symbolic—it surely has little practical or intellectual force today"); Ginsburg, *supra* note 36, at 994–95 ("[T]he differences between the U.S. and French copyright systems are neither as extensive nor as venerable as typically described.").

law jurisdictions show the effect of the fork: authors and artists have additional, non-pecuniary mechanisms for control over their works.[276] Political and social changes initiated this fork. The changes evolved, and, along with later interpretations of these changes, embody today's conventional wisdom for the emergence of author-protecting moral rights.[277]

The political and social changes initiating moral rights start with the French Revolution, and, as a result, their early history is a story about events in France.[278] French law provides the strongest expression of moral rights.[279] In association with the French Revolution, the law of France reacted against censorship by the French Crown with a new copyright code, eliminating the royal prerogative basis for rights in creative works. Under this new code, French courts would develop the *droit moral* to protect the author's or artist's relationship with her work.[280] Later, and for many years, French courts evolved these rights: "Moral rights did not seem to have been in the destiny of French intellectual property law, but instead the rights resulted from practical encounters in the courts."[281] Eventually, after extensive judicial development, the French codified these rights in statutory law in 1957.[282] Thus, moral rights emerged in France along with momentous historical change and developed from these changes and beyond.[283] In varying forms and following the

---

[276]*See* GOLDSTEIN, *supra* note 20, at 283–84 (discussing generally moral rights). This becomes less true over time as international and regional harmonization pressures bear on local copyright systems, causing, for example, England to enact a degree of protection for moral rights. PHILLIPS ET AL., *supra* note 16, at 58.

[277]*See* GOLDSTEIN, *supra* note 20, at 9 (reporting that latter-day commentators expanded Immanuel Kant's work connecting literary creation to personality of author); Ciolino, *supra* note 26, at 939–40 (identifying monist theory of moral rights with Kant, but identifying dualist theory with Georg W.F. Hegel). I do not challenge this conventional story for the emergence of moral rights.

[278]*See* Peeler, *supra* note 29, at 427 (tracing "the judicial origins of French moral rights").

[279]*See* Halpern, *supra* note 267, at 72 (noting that France's implementation of moral rights are "most expansive"); Ilhyung Lee, *Toward An American Moral Rights In Copyright*, 58 WASH. & LEE L. REV. 795, 803 (2001) ("Within the international community, France is the undisputed champion of authors' moral rights . . . .").

[280]*See* Peeler, *supra* note 29, at 427 ("[T]he Revolutionary government that enacted the first law was politically motivated to reject all symbols of the centralized power and absolute control that the French monarchy had previously exercised over authors and artists.").

[281]*Id.* at 432 (noting that judicial development arose from a legitimate need to determine the meaning of intellectual property law in nineteenth-century France, but that the development was colored by the culture's "growing adoration of creative genius [which] impelled the judiciary to fashion principles of moral justice as an important tenet of the law of authors' rights").

[282]*Id.* at 426.

[283]*See generally id.* at 449–54 (tracing changes in French law through nineteenth century and beyond).

emerging trend in France, moral rights also developed in other major civil law jurisdictions.[284]

Commentators link moral rights to theories of personality in property.[285] That is, the idea that property—the right to exclude something about an item to some degree—attaches to a physical object or intangible item by a person imbuing their will, their personality, to that item.[286] For example, in the sculptor example for the right of integrity, the sheer act of creating the statue, because it is an expression of creative energy manifesting in the object, gives the sculptor a metaphysical basis to claim moral rights in the work—to have some measure of control to ensure that the work stays as the sculptor originally intended.[287] By doing so, the sculptor ensures that the work continues to

---

[284]Halpern, *supra* note 267, at 72. Halpern summarizes the variance in moral rights among civil law jurisdictions as follows:

> Within this group of different cultures, there is wide variation in the scope and duration of moral right. While sharing much, the ambit of protection accorded by France, Italy, and Germany varies considerably as does that of the many different signatories to the Berne Convention. The duration of moral right ranges from the lifetime of the creator to perpetuity; different parts of the bundle are protected, or left unprotected; interpretation may be broad or narrow. In short, beyond the most general principles, there is no universal set of moral right constructs applicable even as among the Civil Law countries.

Id. (citation omitted).

[285]Kwall, *supra* note 25, at 7–8 ("[T]he moral right doctrine safeguards rights of personality rather than pecuniary rights. The creator projects his personality into his work, and thus is entitled to be free from vexatious or malicious criticism and from unwanted assaults upon his honor and professional standing." (citations omitted)).

[286]Radin, *supra* note 33, at 959–61 (arguing that "personhood perspective" refers to class of property arising from individual's connection to external object such that riddance of object would occasion felt loss, but contrasting this class of property with external items held purely for instrumental reasons). Radin notes that such a personhood conception for property requires, to avoid total subjectivism, some concept of a "person" to calibrate which extensions of a person's will to an external object should occasion property rights. *Id.* at 961–62, 973–74, 977–78, 986.

[287]Liemer, *supra* note 17, at 44. Liemer expresses this point as follows:

> Moral rights seek to protect the artist's creative process by protecting the artist's control over that process and the finished work of art. If artists feel more secure about the treatment they as creators and their creations will receive, they are more likely to create. Recognizing moral rights is one way a society can encourage artists to create. While moral rights often may aid the financial interests of the artist, their focus and intent lies elsewhere, in the personal interest of the artist in her work. Indeed, most countries that recognize moral rights also provide a separate set of rights, such as copyrights to protect economic interests in the work.

Id. (citations omitted).

express and embody the sculptor's personality.[288] The world sees and experiences the sculptor's personality through the work. Thus, if the work is mutilated, it can no longer express the artist's personality. Literally, the right of integrity prohibits certain types of modifications; but in light of personality theories of intellectual creation, its deeper meaning comes through: preserving the view presented by the work, and with it the author's or artist's expressed personality.

The other moral rights also fit the personality theory.[289] The author or artist needs to control the first publication or disclosure of the work in order to ensure that when the work leaves the author's domain, it embodies the personality-view desired.[290] Once released, the right of attribution ensures that the original author or artist retains the degree of association with the work under which the author released it. This is often done by name, but could also be under a pseudonym, or be anonymous.[291] The right to withdraw the work upon remuneration also fits the personality theory. If the artist changes the genre or reworks the image, it may be fitting, from a moral rights perspective, for the artist to withdraw from circulation works that clash with a prior era in the artist's development.[292]

---

[288]Netanel, *supra* note 18, at 382. Netanel describes the personality basis for moral rights as follows:

> This personal connection has been variously described as one of artistic reputation, emotional sensibility and dominion of personality . . . . In the aggregate, the [moral rights] serve to enhance the author's ability to determine at all times whether, when, in what manner and form, by whom, and in whose name the work will be presented to the public.

Id.

[289]Personality theory is not the only explanatory vehicle for moral rights. A complementary explanation is that moral rights help society protect and preserve its culture. Authors, artists, and other creators, via their moral rights, assist society to retain works of heritage. *See, e.g.*, Kwall, *supra* note 25, at 15–16 ("[F]ocusing on society's interest in preserving its cultural heritage, when a creator's work is altered after his death, society is the ultimate victim for it can no longer benefit from the creator's original contribution."); *but see* Cotter, *supra* note 17, at 74 (arguing that "endowing the artist with a moral right is a rather awkward method for protecting the public interest in the preservation of art"). Another explanation is that moral rights allow authors and artists to protect their reputation and livelihood. *Id.* at 69–70 (discussing reputation); Hansmann & Santilli, *supra* note 17, at 104–05 (discussing artists' pecuniary interests).

[290]Liemer, *supra* note 17, at 52–54 (arguing that right of disclosure "protects the unique relationship in the arts between person, process, and product" by ensuring that others see work only when authors or artists are ready—when they know that creation process is complete).

[291]Hansmann & Santilli, *supra* note 17, at 130–34.

[292]*Cf.* Liemer, *supra* note 17, at 54–55 ("Arguably, the moral right most difficult for many Americans to fathom is the right of withdrawal."); Hansmann & Santilli, *supra* note 17, at 139–41 (noting that right of withdrawal is much less used than other rights and is limited in important ways, including requirement in some jurisdictions that "a court must determine that the author will otherwise suffer grave moral damage").

Moral rights developed concordantly with their purpose, meaning that limits to the rights developed along with the rights themselves in certain ways and in certain situations.[293] These limits, while in no way uniform across jurisdictions, reflect the tension with the pecuniary conception of copyright, and perhaps signal implicit acknowledgement that the control granted by moral rights, unchecked, could have negative repercussions.[294] Their textual expression narrows their application. For example, the right of integrity does not grant control over all modifications, but only over a "distortion, mutilation or other modification of, or other derogatory action in relation to, the said work, which would be prejudicial to [the author's or artist's] honor or reputation."[295] In a sense, this is congruent with and supports the right of attribution because it relates to the author's or artist's reputation. In addition, moral rights show varying degrees of strength in how they differ among jurisdictions. Differences include their term, and other attributes, such as whether they can be waived or assigned, or the type of relief available.[296]

Originating in France, moral rights are one of the many differences distinguishing the civil law tradition from the common law tradition that developed in England. The English approach to copyright did not embrace moral rights. This explains in part the United States' long-standing hesitancy toward moral rights. The next Subsection elaborates.

2. United States Avoidance

For a variety of reasons, the United States did not originally embrace moral rights. Then the United States managed to avoid adherence to the Berne Convention for the Protection of Literary and Artistic Works for nearly one

---

[293]*Cf.* Michael B. Gunlicks, *A Balance of Interests: The Concordance of Copyright Law and Moral Rights in the Worldwide Economy*, 11 FORDHAM INTELL. PROP. MEDIA & ENT. L.J. 601, 604–05 (2001) (describing American unease concerning moral rights and noting European limits on moral rights). For example, "it is commonly accepted in France that an author's right of respect is not as strong when his work is adapted by a third party than when it is reproduced by the latter." Françon, *supra* note 27, at 82 (citation omitted).

[294]In particular, common law countries influenced the original addition of moral rights to the Berne Convention in 1928 by ensuring that Berne only addresses the rights of attribution and integrity, and by insisting on use of the phrase "honor and reputation" rather than "moral interests" in the right of integrity because the former phrase better tracked common law torts. Gary Lea, *Moral Rights and the Internet: Some Thoughts from a Common Law Perspective*, THE INTERNET AND AUTHORS' RIGHTS 87, 91–92 (Perspectives on Intellectual Property Series, Michael Blakeney ed., 1999).

[295]Berne Convention, *supra* note 28, § 6bis(1). *See* William M. Landes, *What Has the Visual Arts Rights Act of 1990 Accomplished?*, 5 (John M. Olin Law & Economics Working Paper No. 123 (2d Series), May 30, 2001), *available at* http://www.law.uchicago.edu/Lawecon/ (last visited July 13, 2003) (noting that VARA's right of integrity only protects against alterations that injure honor or reputation).

[296]Dietz, *supra* note 28, at 212–19 (comparing various moral rights laws in several jurisdictions); Lea, *supra* note 294, at 101–03.

hundred years.[297] In doing so, it avoided international treaty commitments that would require it to establish moral rights in U.S. law. Just over a decade ago, the United States joined the Berne Convention, but with an underwhelming adherence plan to meet Berne's prescription for moral rights. The pattern continued when the World Trade Organization/Trade-Related Aspects of Intellectual Property ("WTO/TRIPS") regime came into being in 1995. The United States successfully negotiated an exception in TRIPS for moral rights implementation.[298]

Inheriting our copyright tradition from England, United States copyright law never wholeheartedly embraced moral rights.[299] Our system's opposition to moral rights range from ideological to pragmatic. First Amendment freedom of expression traditions arguably conflict with strong moral rights.[300] There is a potential chilling effect from moral rights, in particular the right of integrity, when one seeks to criticize or parody a work in ways that make use of the work in modified form.[301] Moral rights also add friction to the alienability of property and transferability of rights. In a moral rights system, if the work is embodied in physical form, such as a statue, a buyer takes it subject to the possibility of these rights. This reduces the theoretical range of use for the work. Perhaps this prohibits its transfer to a user who would most highly value it if her use would modify the work contrary to the right of integrity. Another reason for our system's hesitancy is that moral rights may require judges to assess aesthetic values.[302]

Pragmatic concerns buttressed these various ideological reasons to oppose moral rights. By giving more control to authors and artists, moral rights would upset the balance established among commercial publishing and distribution

---

[297]*See* Hansen, *supra* note 272, at 586–87. Hansen nicely summarizes the United State's progression toward Berne Convention membership as follows:

> The United States did not provide protection for foreign works for over 100 years. When the United States finally did begin to provide protection, it imposed a requirement that books be manufactured in the United States in order to protect the domestic printing industry. The United States imposed a system of formalities, the main purpose of which seemed to be to throw works into the public domain, including many famous foreign works. It just recently joined the Berne Convention, and did so only because other nations told it repeatedly, "If you are going to preach the religion [of high protectionist copyright], you must join the Church."

Id.

[298]*See* Lea, *supra* note 294, at 94.

[299]*See* Halpern, *supra* note 267, at 65–69.

[300]Kathryn A. Kelly, *Moral Rights and the First Amendment: Putting Honor Before Free Speech?*, 11 U. MIAMI ENT. & SPORTS L. REV. 211, 212–13 (1994); Geri J. Yonover, *The Precarious Balance: Moral Rights, Parody, and Fair Use*, 14 CARDOZO ARTS & ENT. L.J. 79, 92–93 (1996).

[301]Yonover, *supra* note 300, at 103–04.

[302]Paul Goldstein, *Infringement of Copyright in Computer Programs*, 47 U. PITT. L. REV. 1119, 1121 (1986); Robert A. Gorman, *Copyright Courts and Aesthetic Judgments: Abuse or Necessity?*, 25 COLUM.-VLA J.L. & ARTS 1, 2, 10–13 (2001).

interests, and other interests, such as those of the public, and of authors and artists themselves.[303] In addition, some posit that moral rights can harm incentives for collaborative works. For example, they may necessitate additional pre-project bargaining to obtain waiver for the moral rights, assuming that they can be waived in the applicable jurisdiction.[304] Finally, moral rights can be viewed as less applicable to functional works, because they have no artistic, personal, or cultural heritage.[305]

While ideological and pragmatic reasons kept moral rights mostly out of United States law, two contrary developments arose. First, in response to the art lobby, several states implemented moral rights for narrow classes of works. By 1990, a number of states had passed laws providing some form of moral rights to works of visual art.[306] Second, international harmonization pressures grew for the United States to join the Berne Convention. The United States finally did so in 1989. In joining Berne, the United States initially argued that, via an amalgamation of rights available under state law, the Lanham Act, copyright law's derivative work right, and other sources, its law substantially complied with Berne's prescription of a moral right of attribution and integrity.

Shortly after joining Berne, Congress enacted the Visual Artists Rights Act ("VARA").[307] VARA implemented a federalized version of the state moral rights laws. Like its state law predecessors, VARA applied only to narrowly defined classes of work.[308] The total effect of these developments, however, still leaves naysayers disputing that United States law provides moral rights as specified in the Berne Convention. Even with state law and VARA moral rights for visual art, and other moral-rights-like provisions found in United

---

[303]Lea, *supra* note 294, at 94.

[304]*See* Landes, *supra* note 295, at 5; Lea, *supra* note 294, at 95–96.

[305]Dietz, *supra* note 28, at 226 (noting that adjustments have been made to European moral rights for certain types of works, such as "computer programs, film works and architecture, where legislators, also of civil law countries, have already introduced some specific provisions that take into consideration the specific, often utilitarian character of those works"); *see also* Lea, *supra* note 294, at 98 (noting that computer programs are exempt from moral rights protection in United Kingdom and covered in limited fashion in France).

[306]*See* Landes, *supra* note 295, 15–21 (noting that nine states passed acts to provide some form of moral rights protection, and surveying some potential economic effects of such legislation).

[307]Visual Artists Rights Act of 1990, Pub. L. No. 101-650, tit. VI, 104 Stat. 5089, 5128–33 (1990) (codified as amended in scattered sections of 17 U.S.C.).

[308]17 U.S.C. §§ 101, 106A(a)(3)(B) (noting that works of visual art includes only paintings, drawings, prints, sculptures, or photos for exhibition, as single works or in limited collections of no more than two hundred, while excluding many other classes of works; and that works must be of recognized stature before their destruction is covered by act). Around the time of VARA, Congress also made buildings eligible for copyright protection under certain conditions. *See* Architectural Works Copyright Protection Act of 1990, Pub. L. No. 101-650, § 702, 104 Stat. 5133 (1990). This is another example of United States law implementing moral-rights-like protection for artists.

States law, the criticisms continue that the United States has only tepidly complied with Berne's Article 6bis.[309]

International intellectual property harmonization pressures have moved away from World Intellectual Property Organization ("WIPO") administered treaties such as Berne to the WTO's TRIPS regime, but without leaving Berne and other WIPO treaties behind. TRIPS requires countries to implement much of Berne, but exempts Article 6bis. Practically, this means that the enforcement mechanisms associated with TRIPS are inapplicable and provide no forum for those harboring the criticisms that the United States does not comply with Article 6bis.

While the United States' aversion to a strong moral rights regime is in part rooted in the common law heritage, England has recently enacted moral rights.[310] England's implementation is in response to European legal harmonization efforts associated with the European Union. It shows, as some commentators have noted, that elements of the two systems are becoming intertwined in many countries' systems as copyright changes in response to international harmonization and technological pressure.[311]

In light of the foregoing, moral rights create two dichotomies, one doctrinal and one historical. First, under the alternative personality theory justification for property-like rights in creative works, as opposed to a pecuniary justification, moral rights create dual systems of rights for authors and artists in some jurisdictions, most notably, France. The second dichotomy is the split among civil and common law systems in embracing or rejecting moral rights. Against this backdrop, and to see how these dichotomies express themselves in perhaps the most powerful of the moral rights, the next Section explores the moral right of integrity in greater detail.

---

[309]*See* Roberta Rosenthal Kwall, *The Attribution Right in the United States: Caught in the Crossfire between Copyright and Section 43(A)*, 77 WASH. L. REV. 985, 987–88 (2002) (arguing for federal adoption of generally applicable right of attribution); *see also* Yonover, *supra* note 300, at 99 n.107 (describing United States support for moral rights post-VARA as limited).

[310]*See* PHILLIPS ET AL., *supra* note 16, at 10 (discussing passage of Copyright, Designs, and Patents Act of 1988 in United Kingdom).

[311]*See* GOLDSTEIN, *supra* note 20, at 9–10; PHILLIPS ET AL., *supra* note 16 (discussing authors' rights in France and Germany). Canada offers both traditional pecuniary copyright protection, and widely applicable moral rights. Jonathan Stuart Pink, *Moral Rights: A Copyright Conflict Between the United States and Canada*, 1 SW. J. L. & TRADE AM. 171, 186–87 (1994).

B. Right of Integrity

Among the traditional moral rights, after a work's divulgence, the right of integrity seems to have the greatest reach.[312] With the right of attribution, it shares an emphasis on reputation.[313] Following a framework used to analyze the French right to respect, the Berne Convention's expression of the right of integrity can be thought to have two aspects which share a condition.[314] One aspect is modifying the work, relating to the phrase "distortion, mutilation or other modification of" in the Berne expression. The second aspect is the work's surroundings: has it been placed in an environment against the spirit of the work? This would relate to the phrase "or other derogatory action." The condition is that whichever aspect is triggered, the result must be "prejudicial to [the author's or artist's] honor or reputation." Recalling the statue example, the subsequent painting triggers the first aspect, assuming that it is prejudicial. For the second aspect, however, more context is needed. Assume that the statue is fine art and is used as a mannequin in a department store—perhaps this would be an "other derogatory action" with respect to the work. Thus, one risks violating the right of integrity if one changes the work, or changes its milieu, by too much.[315]

As with the other moral rights, and rights generally, the right of integrity's other attributes shape its force and effect, such as: whether it is waivable (in whole or conditionally), assignable, inheritable, its duration, and remedies.[316]

---

[312]The right of integrity has the greatest reach because, as a practical matter, the withdrawal right is seldom used and is unknown in many jurisdictions otherwise providing moral rights. Dietz, *supra* note 28, at 204–05. *See* Kwall, *supra* note 309, at 1027–28 (arguing that implementing more comprehensive right of integrity in United States law is more disruptive than implementing generally applicable right of attribution). Its reach is greater than the right of attribution because it theoretically controls a greater range of use for the work.

[313]*See* Mark A. Lemley, *Rights of Attribution and Integrity in Online Communications*, 1995 J. ONLINE L. art. 2, ¶¶ 24–28, 41–43, *at* http://www.wm.edu/law/publications/jol (noting interdependence of right of attribution and right of integrity in one's online persona in context of electronic Internet communications).

[314]*See* Françon, *supra* note 27, at 77–78 (describing French right to respect as having one aspect that goes to protecting integrity, that is, respecting current form of work and not modifying it without author's knowledge, and another that goes to protecting spirit of work: that is, author "protests against the environment given to it by a third party").

[315]PAUL GELLER, INT'L COPYRIGHT L. & PRAC., at France § 7[1][c][i] (2000). In addition, some formulations of the right of integrity include violation of that right if the holder destroys the work. *See* Colleen Creamer Fielkow, *Clashing Rights Under United States Copyright Law: Harmonizing an Employer's Economic Right with the Artist-Employee's Moral Rights in a Work Made for Hire*, 7 J. ART & ENT. L. 218, 227–28 (1997) (noting that one aspect of VARA's right of integrity includes prohibition against destroying work, but that this prohibition is limited to "works of recognized stature") (citations omitted).

[316]*See* Cotter, *supra* note 17, at 85–86 (summarizing conclusions as to recommendations of economic analysis for waiveability of right of integrity); *see also* Lea, *supra* note 294, at 101–03 (discussing each of listed attributes).

Jurisdictions, even within civil law systems, vary substantially on these attributes,[317] and a full survey is beyond the scope of this Article and beyond what is needed for a comparison. Two examples should suffice to show the range of variance: the French right to respect and the U.S. VARA right of integrity implementation.[318] The French right is perpetual, inheritable, cannot be assigned or waived, and provides damages or injunctive relief.[319] VARA's right of integrity, on the other hand, is non-assignable, cannot be inherited, lasts for the life of the author, can be waived in a sufficiently specific writing, and has remedies similar to pecuniary copyright remedies.[320] That these two examples would lie on either end of the spectrum is not surprising given that France is the cradle of moral rights and the United States is a recent tepid entry to jurisdictions having some form of moral rights.[321] The right of integrity's various incantations illustrate the law's adaptive capabilities. The right's implementation in each jurisdiction had to fit the greater legal backdrop of pecuniary copyright as well as economic and cultural influences.

In both cases, the right asserted—the right to respect in France, the right of integrity in the United States—is separate from pecuniary copyright. Both sides of the dualist model are unique and independent rights. Despite their independence, however, they share a more fundamental commonality: they create rights that "run" with the work, or run with the object in which the work is embodied. For pecuniary copyright, these rights link principally to ownership but are limited by doctrines such as first sale when a work is embodied by fixation, or limited in other situations for policy reasons, such as

---

[317]*See* Cotter, *supra* note 17, at 13–15 (collecting number of formulations for right of integrity and giving examples of modifications to works causing violation of right).

[318]While my illustrative example is for the right of integrity, many of these attributes will be the same for the right of attribution in each system. That is, the French right of attribution will vary dramatically from the VARA right for these attributes, while sharing similarities with the French right to respect.

[319]*See* Dietz, *supra* note 28, at 207, 212–13, 217.

[320]17 U.S.C. § 106A(d)–(e). The waiver provision requires a written instrument that shall "specifically identify the work, and uses of that work, to which the waiver applies, and the waiver shall apply only to the work and uses so identified." § 106A(e)(1). VARA's right of integrity is also limited in its applicability to buildings, where the building owner has power under certain conditions to remove the work absent a waiver even if removal causes mutilation of the work. §§ 106A(a)(3), 113(d). VARA treats violations of sections 106A and 106A(a) as copyright infringement for purposes of the remedies chapter of the Copyright Act, except that the criminal penalties for copyright infringement do not apply. §§ 501(a), 506(f). Finally, it is interesting to note that the VARA rights persist for the life of the author when pecuniary copyright, that is, the rights of section 106, persist for the life of the author plus seventy years. § 302(a).

[321]*See* GOLDSTEIN, *supra* note 20, at 291–92 (describing trends generally as to waiver and other limitations among jurisdictions).

fair use.[322] The limitations notwithstanding, pecuniary copyright "runs" with the work in the sense that the copyright owner (not necessarily the original author) can exercise some control over the holder's use of the work—such as prohibiting reproductions.[323]

Moral rights also run with the work. They do so in a more direct way. For example, in VARA, the right of integrity is limited only by fair use in 17 U.S.C. § 107, whereas the pecuniary copyright rights of section 106 are limited by sections 107 through 122.[324] Moral rights also stay with the author, not the copyright owner. This creates the possibility that a work's owner may have to answer to two parties depending on what she does with the work. In the sculptor example, if the work's holder creates duplicates of the now-painted statute, and the sculptor has assigned her copyright to a third party, the holder faces a potential right of integrity violation suit from the sculptor, and separately, a potential reproduction right infringement suit from the third party.

Pecuniary copyright creates rights that attach to software, that in effect "run" with the code. Moral rights, in jurisdictions where they apply to software, also follow the code. In both cases this means, most fundamentally, that a contract or other rights-regime is unnecessary to enforce these rights as the work, or code, transfers down a chain of distribution. The next Section elaborates on how this characteristic, and moral rights in general, apply to software.

C. Moral Rights in Software

The moral rights that run with software are often less powerful, if present at all in a jurisdiction, than moral rights for other copyright subject matter. They can be problematic for software and for the open-source approach. By their very definition, they attach to each copy of the computer program, that is, to the instructional composite. A violation could occur through modifications to the source code instructional composite or by direct modifications to the object code.[325] In either case, modifications of the software that violate the

---

[322]Although principally the copyright owner is empowered to enforce the rights, in certain situations exclusive licensees can also enforce the rights they have been licensed. *See* NIMMER & NIMMER, *supra* note 55, § 12.02[B].

[323]17 U.S.C § 106(1) (2000). *See also* Margaret Jane Radin & R. Polk Wagner, *The Myth of Private Ordering: Rediscovering Legal Realism in Cyberspace*, 73 CHI.-KENT L. REV. 1295, 1312–13 (1998) (discussing contracts that "run with" an object and how these devices can be used to extend or reduce the baseline intellectual property rights upon which they are based).

[324]17 U.S.C § 106 (2000).

[325]For example, many software packages list the names of key developers in the first window or screen displayed. The object code is directing the computer operating system to display the screen. Traditionally, it was a relatively straightforward exercise for a software expert to excise a name from the list, or perhaps replace one name with another, all the while operating directly on the object code.

right of integrity are analogous to painting one statue purple when the sculptor has produced many unpainted copies of her statue. The technical difference is that copying the software is much easier. But, generally put, there is no legal difference if the changes in both cases mutilate, modify, or distort the work in a way detrimental to the author's or artist's honor or reputation.

1. Attenuated Implementation and Coverage

Even in civil law jurisdictions, moral rights in software are attenuated.[326] In other jurisdictions, such as the United States, they are unavailable for software and computer programs. The reasons for this are many and, on the whole, reasonable. First, software's dual character as both an expressive and functional work implies that moral rights are less imperative. In the prevalent personality-theory basis for moral rights, it is the expressive aspect of a work that embodies the creator's personality. The intuition is that if software is primarily or even substantially functional, there is less need for a right of attribution or integrity.[327] Second, moral rights developed for classic copyright subject matter. Given the questions in the early years of computing as to whether software was a literary work protected by copyright, it seems prudent to not endow software with additional rights.

Third, like other digital works, software is inherently more malleable than traditional works fixated in physical form. Computer programs are designed to be modified, or easily modifiable. A right of integrity, where the author can govern modifications, would be counterproductive to the sequential and successive processes used to develop software.[328] In almost all cases, if the software is continually used, it will continue to need modifications. This is the well-known commercial practice of releasing new versions of software. The right of attribution is also problematic for digital works—it is difficult to keep

---

[326]*See* Lea, *supra* note 294, at 98 (discussing limitations on moral rights in United Kingdom and France).

[327]*See* GELLER, *supra* note 314, at France § 57[2][a] (2002) (noting that need seems less urgent to apply French right to respect to works, such as software, which leave less imprint of author's personality); Fakes, *supra* note 266, at 609 (arguing that for moral rights in software "serious doubt exists as to the suitability of their application to works frequently produced collectively, having a technical, industrial or commercial character and subject to successive modifications" (quoting Commission of the European Communities: Green Paper on Copyright, COM(88)173, final at 197)).

[328]My thanks to Mark Lemley for suggesting this point, especially in the case of the traditional moral right of integrity as it might apply to software.

an attribution fingerprint on the work as copies of it propagate across the Internet.[329]

Moral rights in software are attenuated in two ways, jurisdictionally, and by limits to the rights themselves. Some jurisdictions that provide moral rights in other copyright subject matter do not apply them to software. The United States is an example of this case. Other jurisdictions provide moral rights in software but clip the rights. For example, France provides a right of attribution for software, but its right to respect in software explicitly specifies that modifications for use and debugging are allowed.[330] The remaining jurisdictions are those whose moral rights are specified equivalently for software and other copyright subject matter. Examples here are Canada and Italy.[331]

Even in their attenuated form, moral rights, and specifically the right of integrity in its classic form, can present problems for open-source software licenses. I cover these problems in the next Subsection. In the next Part, however, I turn this situation on its head to argue that the right of integrity might enable an additional claim against one who violates the license terms that implement the open-source approach.

---

[329]At least with current and past technology, keeping an attribution fingerprint on a digital work such as source code required the recipients in a chain of distribution to honor the posted attributions. Many recipients had the technological capability to modify the attributions. Within an open-source software project, however, there is usually a technological barrier to misattribution because the source code is submitted to, or checked in and out of, a repository, such as a source code control system ("SCCS"). Thus, one who checked out some code from the SCCS, revised it, stripped the attributing information, and resubmitted it to the SCCS would be easily discovered. Another technology that in the future may increase the permanency of attribution information is Digital Rights Management ("DRM") or Digital Rights Expression ("DRE") technology. These technologies envision a system that automatically controls the uses of a copyrighted work according to the rights granted or expressed in information carried along with the work. *See* Symposium, *The Law and Technology of Digital Rights Management*, 18 BERKELEY TECH. L.J. 697, 715, 732–33 (discussing DRE and DRM and differences between two); *see also* Thomas C. Greene, *MS to eradicate GPL, hence Linux*, THE REGISTER, *at* http://www.theregister.co.uk/content/4/25891.html (June 25, 2002) (positing conflict between newly announced DRM technology from Microsoft and GPL open-source license).

[330]*See* GELLER, *supra* note 314, at France § 7[2][a]; *see also* Lea, *supra* note 294, at 98 (noting France's allowance of computer program debugging).

[331]*See* David BENDER, INTERNATIONAL COMPUTER LAW §§ 7.09, 7.33 (2002) (discussing moral rights for computer programs in Canada and Italy); *see also* GOLDSTEIN, *supra* note 20, at 284 (discussing Canada's law); PHILLIPS ET AL., *supra* note 16, at 17 (discussing Canada's law).

2. Discord with the Open-Source Approach?

The open-source approach[332] seeks to ensure that source code is always available and royalty-free, regardless of the software's end use. Source code availability serves a variety of purposes, one of which is allowing others to modify the software to suit their unique needs. Although many members of the open-source community might disfavor certain uses of their software, out of necessity the open-source approach must be disinterested in the end users' ultimate use. In other words, to be most effective, the approach must not discriminate against those who would put the software to uses which the developers would not. Thus, for example, open-source developers might be heard to complain that they do not want their software used in the engineering process for producing nuclear weapons, or in biotechnology agribusiness labs creating genetically modified organisms ("GMO"). But imposing such restrictions in a license, while perhaps satisfying in the instance, in the aggregate is counterproductive to the aims of the open-source approach. This is generally recognized in the community. For example, the Open Source Initiative's OSD certification program specifically requires that a license not discriminate as to use.[333]

Most developers in collaborative open-source projects are working with source code carrying an interlocking web of copyright-based license permissions.[334] Thus, these developers do not control the copyright in all of the code, and, as a result, they are typically unable to privatize the project if they decide that an end use was unacceptable. They do not own the copyright in enough of the project to privatize it all. Moreover, they have likely modified and redistributed, as well as used, the software under the license granted by other copyright owners whose contributions appear in the software. This binds them to the license terms.

Assume for the moment, however, that a single developer owned the entire copyright in the project. Further assume that this developer was highly dissatisfied with the use of the software by a certain group of end users, GMO biochemists. This developer would have the power to discontinue use of the open-source approach—in effect, privatizing the project. Prior licensees

---

[332]Using the GPL as the paradigmatic example of the open-source approach, it is a generally applicable license that requires, as a condition to use the software, that one who takes the software can use it, modify it and redistribute it if she conforms to the following: (1) makes the source code available, (2) does not charge royalties for software use, (3) propagates the same terms for redistributed or modified software, (4) includes notice of the GPL terms, (5) attributes modifications to the maker, and (6) disclaims warranties and liabilities. *See supra* text accompanying note 102 (discussing GPL).

[333]*See* OSD 1.9, *supra* note 220, §§ 5–6 (prohibiting discrimination against persons or groups, or against fields of endeavor, if license is to meet OSD certification).

[334]*See supra* text accompanying notes 102–106 (explaining how ongoing contributions to open-source software might function).

and end users could presumably continue their use under the previously granted license. But this privatizing developer would at least be able to limit the use of her future development efforts. She might even rerelease future versions of the software under a discriminatory open-source software license—one that implemented the open-source approach but did so only for certain types of end use.[335] Assuming that her programming contributions were important to the software and that any user would prefer using the new version, she would have implemented her goal of excluding the GMO developers from further use of the evolving software. Of course, the developer could also simply stop working on the project, but this may be undesirable due to the developer's investment in the project and her ongoing satisfaction from her participation in the project. In this scenario the developer would have effectively used the copyright power to add a condition to the open-source license: don't use the software to help create GMOs.

Moral rights, specifically the right of integrity, create the possibility of a similar situation and control mechanism but with important differences. One difference is that any programmer could exert the right without holding all the copyrights. Unless her contributed code is so minor that it could be easily excised from the project, a programmer could hold-up a group of users with a claim that the group's modifications, or even mere use, violated her right of integrity in the software. Such a claim would obviously have disruptive effects in the developer community associated with the project.[336] Another difference is that it is unusual and unlikely for a single developer to own all copyrights in an open-source project, whereas a programmer will hold her right of integrity in her source code if the applicable jurisdiction provides the right. Thus, a right of integrity in software might provide the opportunity for a programmer or group of programmers to bypass the hospitable-to-modifications sharing foundation established by the open-source approach. The disgruntled

---

[335]Of course, an open-source license that excluded GMO biochemists would not qualify for certification under the Open Source Initiative's Open Source Definition. OSD 1.9, *supra* note 220, §§ 5–6.

[336]A similar disruptive effect can result from allegations of copyright infringement such as those involved in a suit filed by SCO in March 2003 against IBM, alleging that IBM incorporated software SCO licensed to IBM under confidentiality agreements into the Linux source code. *See* Steve Lohr, *No Concession From I.B.M. in Linux Fight*, N.Y. TIMES, June 14, 2003, at C1 (describing suit and noting concern caused by suit among corporate technology buyers); SCO Files Suit Against IBM, *at* http://www.sco.com/ibmlawsuit/ (last visited May 23, 2003) (plaintiff's Web site describing suit and providing links to documents SCO filed). Later, in developments related to the IBM dispute, SCO "announced plans to seek licensing fees potentially totaling billions of dollars from users of" Linux. *See* David Bank, *SCO Announces Plans to Seek Licensing Fees from Linux Users*, WALL ST. J., July 22, 2003, at B5, *available at* 2003 WL-WSJ 3974680 (also noting that "SCO has retained David Boies . . . , who played a major role in the government's antitrust case against Microsoft"). The move against users is related to SCO's copyright infringement suit against IBM, alleging that IBM in fact incorporated SCO's code into Linux without permission. *Id.*

programmers could wield the right of integrity to govern modifications to the software when the open-source license specifically permits all modifications. Under a dualist conception of moral rights, where the rights are separate from pecuniary copyright,[337] this is at least a possibility.

A right of integrity assertion in open-source software would be different and potentially more detrimental than a fork in an open-source project. With a fork, the disgruntled group simply strikes their own, new path with the software. Under the open-source approach, the mere act of forking gives them no power to object to anyone else's modifications. Indeed, if the forking group's software turns out to be better, the jilted original developers are free to incorporate it back into their project.[338]

One of many nuances to this analysis is whether the right of integrity can be waived.[339] If the jurisdiction's moral rights statute allows waiver in a sufficiently broad fashion, then open-source licenses could require such a waiver, or existing licenses might be read to implicitly waive assertion of moral rights.[340] Another nuance is that, in some jurisdictions, moral rights are

---

[337]*See* GOLDSTEIN, *supra* note 20, at 291 (noting that "in most countries an author's moral rights are doctrinally separate from his economic rights"). Goldstein states that this is illustrated by the "opening phrase of Berne Article 6*bis*(1) guaranteeing the rights of attribution and integrity[:] '[i]ndependently of the author's economic rights, and even after the transfer of said rights.'" *Id.* (citation omitted).

[338]This is another example of how the open-source approach might encourage collaboration. If the jilted original developers decide to incorporate the renegade forking group's code into their project, it suggests a truce. In other words, when the source code and software can be fully examined, the possibility of better solutions for portions of the software arising from another group, channels all participants toward collaboration in order to most effectively combine the best efforts from otherwise partitioned groups.

[339]*See* GOLDSTEIN, *supra* note 20, at 291 (noting that, despite French approach to contrary, moral rights in most countries last "no longer than the author's economic rights . . . and may be subject to waiver"). A number of other issues could bear on the situation I have hypothesized, among the most prominent, I would argue are, choice of law and choice of forum issues. If the software in which the right of integrity is being asserted is used in another non-software-moral-rights jurisdiction, which law applies? This will be related to the choice of forum where the action is brought. These issues are beyond the scope of this Article, but they illustrate the increasing complexity that will bear on the open-source approach as it continues to expand internationally.

[340]These various possibilities for waiver depend on the requirements to waive the right of integrity in the relevant jurisdiction. For example, in VARA, to be effective a waiver must meet the following stipulation:

> [R]ights may be waived if the author expressly agrees to such waiver in a written instrument signed by the author. Such instrument shall specifically identify the work, and uses of that work, to which the waiver applies, and the waiver shall apply only to the work and uses so identified. In the case of a joint work prepared by two or more authors, a waiver of rights under this paragraph made by one such author waives such rights for all such authors.

17 U.S.C. § 106A(e)(1).

perpetual and inheritable.[341] This raises the specter that a contributing programmer's heir would assert the moral right with a different sensibility than the programmer might have exercised.

Thus far, my account of moral rights in software is a story only of the trouble that these rights might cause and how they have accordingly been attenuated. A postulated case for moral rights in software relying on personality-theories of property would argue that a programmer endues her personality in the expressive elements of the software. Programmers can be heard to describe some source code as elegant or beautiful.[342] Source code programming languages allow for flexibility of expression, but much less so than in traditional human languages. So, while there can be elegance, layered complexity, economy of expression, and clever construction in source code, it does not have poetic style, characters, double meanings, or turns of phrase in any degree like traditional copyrightable literary work.[343] An exception is that the comments in the source code might have these characteristics. Moreover, in traditional literary works, sometimes the originality lies in breaking the rules of grammar and syntax. With source code, the rules must be followed. If they are not, the compiler will see errors in the source code, and it may fail to generate the object code version of the instructional composite, or may generate object code that does not perform the intended function or has unintended consequences.[344] Thus, when programmers speak of beautiful or elegant software, these complements principally reach to clever and effective use of the programming language by marshalling, combining and invoking the abstractions and mechanisms the language provides. This, in part, makes a case

---

The VARA waiver requirements are specific to a degree such that if they applied to software, it would be challenging to write a generally applicable waiver to be incorporated into an open-source license covering all uses of the software. If the waiver requirements were less stringent, in particular eliminating the signed writing requirement, an open-source license might be able to incorporate a general moral rights waiver.

[341]Most notably, in France. *See* GELLER, *supra* note 314, at France § 7[3].

[342]*See* Richard A. Danner, *Redefining a Profession*, 90 LAW LIBR. J. 315, 350–51 (1998) (discussing and quoting from David Gelernter's *Machine Beauty: Elegance and the Heart of Technology*, on possibility of beautiful and elegant software).

[343]Some application areas of software and computer science push the limits of my assertion comparing software to traditional literary works, most notably artificial intelligence systems, and gaming software, systems, and environments. However, for the vast majority of commercial software, my asserted contrast holds true.

[344]Recall that the compiler is a special program that translates the source code instructional composite into the object code instructional composite that the computer can execute directly. Unlike an expert human translating German into Japanese, who can likely correct for a broad range of grammatical or meaning ambiguities in the German source, the compiler's ability to correct is more limited. Typically, the compiler can only flag problems for the programmer in a post-compilation report. It usually classifies problems by severity. Thus, some problems may prematurely abort the compilation process, while others may allow it to finish, but (due to errors the programmer left in the source code) result in object code that does not fulfill the intended goals.

against moral rights in the source code instructional composite but leaves open the question of the object code and the computing result.

Some software applications are graphics related and as such their object code may generate audio-visual displays—a computing result that may qualify for copyright protection, not as a literary work, but as an audio-visual work.[345] These works are more directly analogous to traditional visual arts copyright subject matter. As such, perhaps the case for moral rights in these works is stronger than for source code.[346]

The functional nature of software and the source code instructional composite argue against extending moral rights to software. Despite this, some jurisdictions provide such rights, often in attenuated form. The full form is reserved for traditional copyright subject matter, such as visual art and literary works. The visual arts are at the center of traditions that spawned moral rights protection through events in Europe's civil law jurisdictions since the time of the French Revolution. Even the United States has joined the moral rights movement for narrowly defined classes of visual arts. In the past, moral rights cleaved a clear dichotomy among the world's jurisdictions—civil law jurisdictions provided the rights and common law jurisdictions did not. These lines have blurred under international harmonization and other pressures. Harmonization, however, has been minimal with respect to moral rights.[347] Thus, to the extent that moral rights reach software, a greater degree of nonuniformity may wait there for the open-source approach, risking perturbations to the approach.

Despite these differences, from a deeper perspective, an artist asserting a right of integrity in a statue shares something with an open-source programmer seeking to keep her software free, sharable, and available with source code. They share a reputational interest in the respective forms they seek to preserve, and a desire to preserve their work for a greater community. From these parallels, in the next Part I discuss three implications apparent from the comparison.

---

[345]ROBERT P. MERGES ET AL., INTELLECTUAL PROPERTY IN THE NEW TECHNOLOGICAL AGE 407, 958 (3d ed. 2003).

[346]Although the similarity of some computer-generated audio-visual works with traditional visual arts may suggest moral rights protection, other factors may counsel against it, such as the frequent difficulty of identifying a single creator of such works. *See* Lea, *supra* note 294, at 95–96.

[347]*See* GOLDSTEIN, *supra* note 20, at 291–92 (pointing to different levels of moral rights accommodation in civil and common law systems); *see also* Doris Estelle Long, *"Globalization": A Future Trend or a Satisfying Mirage?*, 49 J. COPYRIGHT SOC'Y U.S.A. 313, 319–20, 353–55 (2001) ("present harmonization efforts in areas such as . . . moral rights demonstrates that much [intellectual property right] harmonization is a mirage"); *see also* Hansmann & Santilli, *supra* note 17, at 97–98 ("[W]ithin individual European countries there is often considerable controversy about the precise interpretation to be given existing statutory and decisional law concerning artists' moral rights.").

V. COLLABORATIVE INTEGRITY FOR OPEN-SOURCE SOFTWARE

The open-source movement uses a nascent approach combining copyright law and generally applicable licenses to grant conditional permission for others to use, modify, and redistribute software. The conditions implement the approach, requiring source code availability, no royalties, allowed modifications and redistribution, and reapplication of these same conditions on redistribution. These conditions create a parallel impression compared to the right of integrity: both techniques control the view that a work presents. The open-source license demands a view with source code. The right of integrity demands a view that preserves the author's or artist's personality as expressed in the work.

There are three consequences in this comparison. First, that the comparison teaches a better understanding of the open-source approach. Second, that jurisdictions providing a right of integrity in software, which creates some risk for the open-source approach,[348] also may give an additional basis for programmers to enforce the open-source conditions. Third, that the law should evaluate an altered approach to protecting open-source software inspired by the dualist nature of the moral right of integrity. I sketch an altered model suggested by the traditional civil law author's moral right of integrity, but modified for the collaborative nature of software development: "Collaborative Integrity" for open-source software. The following sections address each consequence in turn.

A. Comparative Implications and Insights into the Open-Source Approach

If, metaphysically, it can be said that an author or artist embodies her personality in her work, then it can similarly be said that an open-source programmer endues her software with an expression of personality when she demands that it be available with source code and be freely sharable. This assertion can be appraised in several ways by examining reputation, values, characteristics of the work, external effects, and effects on the author, artist or open-source programmer.[349]

---

[348] *See infra* Part V.B (discussing problems right of integrity poses for open-source approach).

[349] I ground my comparison in these five metrics because they provide the relevant context for a "consistent character structure" model of personality. *See* Radin, *supra* note 33, at 963–64, 965–68 (discussing four potential models of personality and describing "consistent character structure" model as ability to project continuous life plan into future, where one's consistent character structure integrates interpretations of past and plans for future).

1. Reputation

Reputational interests link the right of attribution and the right of integrity. Both rights help the author or artist preserve their reputation. The right of attribution ensures that the creator is named when she should be and is not named when she should not be.[350] The right of integrity governs modifications to a work when the modifications will not only reflect poorly on the author or artist, metaphysically damaging the expressed personality in the work, but also when the modification may harm the reputation accruing to the artist from the work.[351]

Strikingly, together these two moral rights bracket the variations among open-source software licenses. As an example of a minimally restrictive open-source license, I have previously discussed the Apache license. At the other end of the continuum is the GPL, which is the most "controlling"—meaning that it has the greatest reach and ambition to ensure that the open-source approach applies to the originally GPL-licensed software, any modifications to it, and to software coupled with either of these. The Apache license's primary requirement is attribution. It allows any use of the source code and software as long as attribution is proper and other notices are posted.[352] The GPL, on the other hand, corresponds to the right of integrity: both seek to control the view that a work presents. The GPL also has conditions similar to the right of attribution.[353] The correspondence among the moral rights and the open-source licenses is illustrated in the table that follows. This table includes the OSD as a midpoint approach between the GPL and Apache licenses.

---

[350]My use of "attribution" in this context is broad. For example, the Berne Convention associates attribution with claiming authorship in the work, but does not explicitly speak to situations where the author has been improperly associated with a work. Berne Convention, *supra* note 28, § 6bis(1). Commentators have described these two possibilities as two aspects of the right of attribution, the positive and negative aspect, and noted that jurisdictions vary as to whether they explicitly recognize both aspects in their right of integrity implementation. Hansmann & Santilli, *supra* note 17, at 130–36.

[351]*See* Hansmann & Santilli, *supra* note 17, at 102–03 (noting one's personal reputation as one reason author may wish to maintain right of integrity in their work).

[352]*See* Apache License, *supra* note 141.

[353]*See* GNU, GPL, *supra* note 99, § 2(a) (requiring notices).

Issue	Apache	OSD[354]	GPL
Right of Attribution	correspondence	correspondence	correspondence
Right of Integrity	n/a	partial correspondence[355]	correspondence

Table 2

Correspondence Among Certain Moral Rights and Open-Source Software Licenses

Thus, deeply similar reputational considerations pervade both open-source software and moral rights. Reputation is the effect of one's exposed personality over time. It is the flip side of the expressed personality embodied in a creative work or freely shareable source code. Commentators have noted the effect of reputation in each context.[356] Artists and authors can use moral rights to protect, promote, and enhance their reputation. The right of attribution ensures proper identification with a work. It helps the viewer to associate what she finds in the work with its creator. The meaning, impression, and assessment the viewer takes from the work, along with any preconceptions, forms her opinion of the work and its creator. When she expresses her opinion to others, she contributes to the ongoing construction of the creator's reputation. Thus, through its viewers, a work infuses an artistic reputation into the world.[357] The right of attribution helps ensure that this infusion is properly tagged.

The right of integrity, by governing modifications, helps ensure that the infusion is the proper one the artist set to the work originally—not a false reputation-carrying infusion resulting from someone else's modification of the

[354]Recall that the OSD is not a license, but rather, it is a specification for a certification system operated by the Open Source Initiative to classify open-source licenses. *See supra* note 168 (defining OSD as "a set of guidelines" managed and promoted by OSI). *See also supra* note 261 (noting questions about the GPL and guidance offered by OSD).

[355]The OSD has partial correspondence with the right of integrity because it merely *allows* a qualifying license to require that redistributions reapply the same license terms. But, the OSD specification, unlike the GPL, does not *require* that the open-source license do so. Thus, the OSD does not demand the reapplication provision. Because the OSD makes the reapplication provision permissive, it does not guarantee that modified redistributions of the software will continue to carry the same licensing power to control the view presented by the work. *See* OSD 1.9, *supra* note 220, § 3 ("The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.").

[356]*See* Hansmann & Santilli, *supra* note 17, at 102–05 (noting traditional justification for moral rights stemming from harm to artist's personality embodied in work, and noting effects of aggregate and temporal reputational effects for author's work: "[The] works we label 'art' commonly involve important reputational externalities, thus giving both the artist and others an unusually strong interest in protecting the integrity of individual works.").

[357]*See* Hansmann & Santilli, *supra* note 17, at 104–06.

work. The right of integrity helps the author or artist preserve the view the work presents—the view it had when they originally divulged it.[358] Thus, the work's intrinsic contribution to the author's or artist's reputation is held constant by the right of integrity.[359] Extrinsic factors may change, such as the work's circulation, popularity, cultural and artistic tastes, or even the author's or artist's own agenda or emphasis. But as long as the work exists, the view that it presents endures as long as the right of integrity is available and enforced. If an author or artist has an evolving or even radically changing motif, she faces a choice about whether to enforce the right of integrity for earlier works. Indeed, she might prefer that the earlier works were no longer in circulation contributing to her reputation if she has re-made her style.

The open-source approach has similar reputation-bearing effects. The license conditions typically call for proper attribution and seek to preserve the source code view of the software. Comments in the source code, as well as other technological mechanisms, denote who wrote the code. Indeed, one of the greatest sins in the open-source community is to modify these attributing comments in order to wrongly designate the original coder. The requirement that the source code be available functions synergistically with the attribution condition. This condition makes the source code viewable so the attribution can be identified.

Other programmers, as well as users, or those merely technologically curious, can examine the programmer's code and make a variety of assessments similar to those made of the artist when viewing a work of art. The use of color or perspective, brushwork, and texture impress the art connoisseur as the data structures, modular design, and skilled use of computing capabilities impress the software expert or hacker. The artist can preserve her work's view with the right of integrity. The open-source programmer preserves her work with a combination of technology and the open-source requirement that source code be available. As a digital work, modifications do not impinge on the original, provided that they are made to a digital copy, which is almost always the case. The copies leave an organized lineage and ancestral copies are

---

[358]The author or artist must actively enforce the right of integrity for it to have the effect of locking in the original view presented by the work. In some jurisdictions authors and artists have a right to withdraw the work in certain situations on equitable terms. *See supra* note 26 and accompanying text (citing sources which discuss author's right in civil law tradition, including right to withdraw). If an artist or author withdrew the work this would diminish or eliminate its contribution to the author's or artist's reputation.

[359]*See* Hansmann & Santilli, *supra* note 17, at 105 (noting that "each of an artist's works is an advertisement for all of the others").

typically available for inspection.[360] Thus, a programmer can earn a reputation not only by the code she originally writes, but also by how well she evolves her code over time. Or, she might earn positive reputation by writing code that is well suited for others to modify and evolve. This last point introduces a discrepancy in the comparison.

Artists may or may not care about their reputation within the greater community of artists for their genre, but open-source programmers on large collaborative projects must and do care whether their code works with other programmer's code. How well their code works in the larger project will contribute to the programmer's reputation. The functional aspect of software, along with the need for code from many programmers to come together as an operating whole, adds this aspect to reputation building in open-source software that is not present to the same degree in traditional arts and literary works. This is not to say that artists or authors do not care about their reputations among their peers, but only to highlight that their reputation among that group does not depend on functional interoperability.[361] Moreover, this is not to say that there is no collaboration among artists or authors, but it is collaboration of a sufficiently different degree as to be different in kind. Software's functional nature requires open-source programmers to care about this additional dimension of their reputation.

The foregoing illustrates that moral rights, developed long ago, and open-source licenses, developed recently, have similar reputation protecting functions. This gives reason to acknowledge, at a level higher than might otherwise pertain, the importance of reputation in the open-source approach.

---

[360]The lineage and traceability for old copies of the code are typically provided by a source code control system ("SCCS"), around which most collaborative open-source software projects are organized. The SCCS is a repository technology that holds the code and helps manage and organize programmer contributions to the project. *See supra* note 204 (briefly describing how SCCS functions).

[361]Of course, the impact on reputation from the need to functionally interoperate is a continuum. While much art lies on one end (minimal impact), and most software on the other, some art, and perhaps contemporary entertainment projects, such as film production, fall in the middle. *See* Margaret Chon, *New Wine Bursting From Old Bottles: Collaborative Internet Art, Joint Works, and Entrepreneurship*, 75 OR. L. REV. 257, 258, 266–68 (1996) (arguing that "authorship practices in networked computer environments result in works that disrupt the distinction between author and infringer and that create a type of access to works (the access of a joint author to a joint work) that is underdeveloped in current copyright doctrine[,]" and describing the Chain Art Project as an example of a new type of work of visual art created on the Internet).

In collaborative software projects, there is some technological partitioning of the software to facilitate interoperability among the contributors' code. These mechanisms include design approaches that partition the source code into modules (or in other ways), relying on layering of functions, or use of software "objects" to assist the partitioning. The mechanisms that define the partition boundaries seek to, among other goals, generate indifference. One module or object should be able to exist and function, perhaps at a lower level, even if other modules or objects are nonfunctional or behaving in unexpected ways.

Moral rights' reputation protecting features are central to the rights' operation. This shows their importance for authors and artists because moral rights in France developed during an era when society perceived a need to protect these groups in order to shed vestiges of the old censorship environment.[362] Moreover, the rights promoted French culture and language by protecting its artists and authors.[363] The fact that all this emerged in the moral rights regime, and that it reemerges in similar form for open-source software, highlights the supportive function played by protecting reputation in both regimes. It also gives reason to view the open-source license as critical conditions that have reputational effects. There is a type of feedback that occurs—protecting reputation makes the producers in each system more likely to be able to obtain the satisfactions available from individual creative activity. This pattern, previously emerged for moral rights, legitimizes its role in open-source software. Similar to the way authors and artists needed to develop reputation, the open-source programmer contributes to collaborative projects in order to earn reputation. Protecting the mechanism by which reputation infuses into the open-source community helps the programmer ensure that any "earnings" she obtains will last. The role of reputation in moral rights should give pause to discounting the role of reputation in open-source software. Finally, it reemphasizes the importance of the rights and the mechanisms deployed to protect it.

2. Values and Beliefs

If reputation is an external pressure for the artist to create, or the open-source developer to contribute code, then each group's values and beliefs are the internal pressure. This is the second way in which I will appraise the assertion that the open-source approach carries and expresses personality equivalently to moral rights.

Personality in the sense used to classically justify moral rights is a metaphysical concept.[364] Even so, values must be a part of what comprises it. Among whatever attributes that the author or artist endues to her work, expression of values must be one. The values appear in the choice of work or

---

[362]*See* Michael P. Ryan, *Knowledge-Economy Elites, The International Law of Intellectual Property and Trade, and Economic Development*, 10 CARDOZO J. INT'L & COMP. L. 271, 291 (2002).

[363]*See* Hansmann & Santilli, *supra* note 17, at 106 (noting that "community benefits [from art] have the character of a public good, [and thus,] the current owner of the artwork has insufficient incentive to protect them by protecting the work itself"). Thus, moral rights help protect the character of art's public good, and, in parallel fashion, the open-source approach helps protect the foundation for open-source collaboration—which has produced public good software.

[364]*See* Elliot C. Alderman, *Resale Royalties in the United States for Fine Visual Artists: An Alien Concept*, 40 J. COPYRIGHT SOC'Y U.S.A. 265, 282 (1992).

subject matter, its medium of expression, as well as the many choices during implementation.[365] For example, a sculptor chooses to create Native American historical figures only in life-size format, sometimes presenting the character with contemporary objects, but always using very dull materials and colors because she believes that this motif will stir a response and carry a message. In addition, an author's or artist's values may include practicing one's passion even without pecuniary benefit. The "starving artist" is a stereotype, but it is a stereotype that illustrates values that put the experience of artistic creation above increased income or economic security. Thus, multiple aspects of an author's or artist's work express a few, or many, of the creator's values.

In addition to the author's or artist's individual values, societal or communal values inform moral rights and approve, at some level, the construct that the creator endues the work with something of herself. The birth of moral rights occurred in an environment where post-revolutionary French society valued art and culture.[366] The society sought to preserve authors' and artists' freedom to create without government interference like that imposed by the Crown before the revolution. The force of this felt societal need is evidenced by the surprising judicial development of moral rights in a civil law system where the statutory code is supreme and judges are not thought to develop law in a common law method.[367] Moreover, the copyright code against which French judges developed moral rights expressed, to some degree, the pecuniary copyright ideology—incentives for production of expressive works by providing a modicum of protection—rather than purely an author's protection rationale.[368] Thus, societal tastes and values that venerated artistic output supported production of that output.

Just as the author's or artist's individual values emanate from her work, so do the values of an open-source programmer who contributes code to a collaborative project or who goes it alone on a solo project. The choice to start or join an open-source project is a value choice linked to the beliefs and motivations of the open-source movement: that source code should be available to show what it teaches; that it should be freely shareable in order to facilitate the first goal and as a matter of principle; and that sharing creativity is a reward unto itself. By contributing code to a project, the open-source

---

[365]*See* Liemer, *supra* note 17, at 43 (discussing how art, regardless of its medium or message, allows "others a glimpse into [the artist's] individual human consciousness").

[366]*See id.* at 41.

[367]Peeler, *supra* note 29, at 433–37, 452–55. Hansmann and Santilli note another out-of-character aspect of moral rights between the civil law and common law traditions:

> [P]atterns of rights that are mandatory under the civil-law regimes of Europe have been forbidden by the common law. This is in strong contrast to the usual relationship between these two legal systems: in general, the common law is far more hospitable to the creation of divided property rights than is the civil law.

Hansmann & Santilli, *supra* note 17, at 96 (citation omitted).

[368]*See* Ginsburg, *supra* note 36, at 1014.

programmer expresses these values. Moreover, the programmer's project choice is a value expressing action. She may choose to work on a graphics application rather than an operating-system hardware driver, because she believes that the open-source approach has critical mass and is doing well in the server operating-system market but needs contributions in graphics to have a chance to develop a presence on the desktop market, where Microsoft Windows products currently rule.[369]

The programmer determines these value-expressing actions within the greater context of the open-source community, a virtual society that both supports the open-source approach by promulgating open-source licenses to protect the foundation for collaboration, and provides the technological and communal collaborative superstructure to develop open-source software for fun and profits of the nontraditional source.[370] Unlike the French development of moral rights, the open-source community cannot necessarily count on courts to develop the rights from a felt necessity of the times. However, the open-source movement can and has promulgated its own generally applicable private law under open-source licenses, and this has a similar effect. Thus, the community is a reinforcing attendant of the values that the programmers express.

These parallels note that value expression has a role similar to that of reputation in seeking to better understand the open-source approach. It makes sense to assign importance to value expression in the open-source approach, because it is the reemergence of an earlier manifested pattern for moral rights and their value-based development and course of use.

3. Source Code Expressing "Personality"

The discussion of reputation and values shows that open-source software can embody and express personality, but it assumes a capacity for software to express personality similar to that of other copyright subject matter. This assumption requires examination, because software has unique attributes as copyrightable subject matter. Despite the differences, in terms of personality expressing capacity, the similarity is sufficiently close to conclude that the open-source approach carries and expresses personality equivalently to moral

---

[369]One example of the efforts to develop open-source software for the desktop market where Microsoft's Windows operating system is dominant is the "Lindows" variant of Linux, which advertises that it delivers "the power, stability and cost-savings of Linux with the ease of a windows environment." *What is LindowsOS?*, *at* http://www.lindows.com/lindows_sales_intro.php (last visited July 31, 2003).

[370]The open-source community includes facilities such as the SourceForge Web portal for open-source projects. It provides a free place for open-source programmers to control and manage their collaborative projects. *See supra* SourceForge, *About SourceForge*, note 107 (highlighting functioning features of SourceForge).

rights, even if traditional closed software does not, or perhaps cannot, because the source code is not available to be viewed.

The expression possibilities in source code are certainly substantial, but are less than that of traditional human languages, because source code, while implemented in a "language," is also a functional tool created to direct, marshal, and invoke computing resources.[371] Programmers may think of source code as beautiful or elegant, but this praise is more about respect and admiration as to how the language tools were deployed rather than beauty in the traditional artistic sense. Even so, when the programmer's deployment choices are exposed in disclosed source code, they carry reputation and values.

The reputational effects are apparent. The source code displays the programmer's skill and knowledge in many ways. Both her coding skill and software design skills are on display.[372] Her knowledge of efficient and clever techniques for data modeling and algorithm deployment is also on display. The choice of project to which she contributes also adds to the impression. Moreover, this is not a static process, evaluated only upon the initial submittal of the programmer's source code. It is dynamic, as the programmer responds to feedback about her software, interacts with other developers while incorporating her software into a larger whole, and evolves the source code based on these inputs and user feedback. Similarly, the source code carries expressions of the programmer's values, sometimes explicitly in the comments embedded in the code, or else implicitly by the choice of where and how to contribute. Software generally may not carry or express a cultural or personal heritage like traditional artistic or literary copyrighted works.[373] If this is a truism, open-source software is the exception—it is unique because it is production activity for non-pecuniary reward (or at least not for direct pecuniary reward). It springs from a value-carrying and value-expressing heritage emphasizing collaborative sharing and peer-review of software's source code.

---

[371]*See supra* text accompanying notes 341–43 (explaining how "[s]ource code programming languages allow for flexibility of expression, but must less so than in traditional human language").

[372]Since many applications can be equivalently developed in a number of different programming languages, even a programmer's choice of programming language signals his or her skills, preferences, and even values and personal expressiveness. Some programming languages are designed to allow greater flexibility of expression, and some are associated with particular ideologies or viewpoints within the open-source community. *See* Lee Gomes, *Two Men, Two Ways To Speak Computerese And Two Big Successes*, WALL ST. J., July 21, 2003, at B1, *available at* 2003 WL-WSJ 3974546 (describing two open-source scripting languages and the key developers behind each, contrasting the characteristics of each language to the characteristics of the key developer).

[373]*See* Drexl, *supra* note 21, at 12–13 (contrasting computer software with traditional artistic and literary works).

4. External Effects

If the open-source approach carries and expresses personality equivalently to moral rights, then some parallelism should exist between the primary external effects of each: a tendency to preserve in some relevant way the works in question. One justification for moral rights is that it helps to preserve a society's artistic and cultural heritage. To the extent authors and artists enforce their rights, in particular the right of integrity, they preserve art and other copyrighted works for society.[374] The sculptor who guards against the painting of her statue by wielding the right of integrity may satisfy her own desire to counter this mutilation, but her efforts also preserve the work in its original form for others to view. To the extent the work is culturally important, her efforts may have important third-party impact.[375] In aggregate, the right of integrity, depending on its attributes (e.g., term and waiver[376]), should create a pressure resulting in preservation of more works than would otherwise pertain. Even a weakly specified right, such as a waivable right of short duration, should have some preservative impact.[377] A strong-form right, such as the French right to respect, with its perpetual term and inability to be waived, should, all else being equal, cause an even greater preservative pressure.

---

[374]*See* Cotter, *supra* note 17, at 36–37, 73–76 (describing the potential protective impact of moral rights on works of art, but arguing that questions exist as to whether this mechanism is optimal). Cotter notes the difficulties

> [E]ndowing the artist with a moral right is a rather awkward method for protecting the public interest in the preservation of art. Perhaps the most obvious cost is administrative. A society that endows artists with moral rights necessarily incurs costs related to the enforcement and administration of those rights, whereas a society that chooses not to recognize them incurs analogous costs only on the rare occasion that someone chooses to attempt to create moral rights by contract.

Id. at 74 (citation omitted).

[375]*See* Cotter, *supra* note 17, at 37 (discussing potential third-party effects of artist/buyer transaction).

[376]Other terms that define the features of moral rights include whether they are inheritable, and their status in what United States law would call a work for hire situation. *See* Fielkow, *supra* note 315, at 220 (describing "clash between the work made for hire doctrine and the moral rights doctrine" where Second Circuit has held that under VARA "art was a work made for hire and therefore fell outside of the statutory protection").

[377]*See* Hansmann & Santilli, *supra* note 17, at 104–07 (discussing effect of rights on different interests). One commentator's attempt to empirically assess the effect of state laws in the United States, which protect moral rights, showed inconclusive results as to the effect of these laws on artists' earnings. *See* Landes, *supra* note 295, at 15–21 (correlating state moral rights protection to various economic indicators, and finding that while there was no effect on artists' earnings, there was "a positive and significant effect on the number of artists living and working in the state"). Landes postulates a possible explanation for the increased artists population density in states with moral rights laws: "the rhetoric surrounding these laws and the prestige of the people supporting them signal to the community at large that art is a highly valued social enterprise. In turn, this creates greater interest in art and a more favorable social environment for artists." *Id.* at 19.

The open-source approach directly contemplates a similar phenomenon. One stated purpose of the movement is to make source code available. Due to the way the open-source approach works, via the reapplication provision (requiring the same license terms to be applied on redistribution of modified versions), the original source code as well as evolving versions of the code remain available for others to view. As a result, the code's lineage is preserved, because the technological systems used to manage the software will preserve successive versions.[378] Even more important, however, is preserving the software from privatization—keeping the open-source software's partial dedication to the public domain vibrant and viable. For the open-source approach, privatization is mutilation that derogates the honor of the programmers who collaboratively developed the project.

Just as the moral right of integrity exists in weaker or stronger forms, depending on the jurisdiction, open-source licenses exert more or less control over the software. Even Apache, the weak-form license primarily providing an attribution-like right, has some preservative effect in the sense of forestalling privatization. Indeed, in conjunction with a well-organized collaborative effort and the benefits of centralizing the functionality, forks in the project have not occurred and competitive offerings via privatization (completely allowed under the Apache license) have not emerged.[379] Thus, there seems to be a preserving effect of even a weak open-source license just as there would be a preserving effect of a weakly specified right of integrity. On the other end of the scale, a strongly specified open-source license such as the GPL eliminates the Apache license risk of privatization. As a result, it will more effectively preserve the source code view of software for all those who desire to modify and

---

[378] *See supra* note 360 (noting that lineage and traceability of code are provided by SCCS).

[379] The Apache license allows any use of the code as long as attribution is maintained and other notice requirements are met. Undoubtedly some Apache code has been privatized in the sense that companies have used portions of the code in traditionally licensed software products, but the entire project has not been privatized in the sense that no significant competing, traditionally licensed product has arisen from the Apache source code base. This preservative effect is driven by at least two preferences of the technologically sophisticated users of the Apache product—often Web site operators and managers. First, they are comfortably familiar with the technical expertise of the open-source developers and prefer the product from that group. Second, they prefer source code availability to enable them to make small customizations. *See* Mockus et al., *supra* note 4, at 318–19 (describing interactions among the core Apache developers and the greater user/minor-developer groups); *see also* Bessen, *supra* note 5, at 15, 17–18 (noting that although "someone could legally use the Apache code to produce a customized closed source product, this has not been done").

redistribute the software.[380] The GPL also eliminates royalties for use of the software, providing further preservation, because the incentives to use the software are heightened by the lowered cost to do so.[381] Greater use and proliferation means greater preservation, in the sense that more users means more potential modifiers and redistributors, all of whom must operate under the GPL.

The parallels between moral rights and the open-source approach are about preserving an important characteristic of each type of work. The parallel is not fully identical. The right of integrity preserves the original form of the work, whereas the open-source approach preserves a particular form for software that allows modifications and collaboration. The difference is that one seeks to prohibit modifications while the other seeks to promote them. This surface difference, however, should not obfuscate the deeper parallelism. Each regime promotes an end result which has, arguably, beneficial external effects for third parties and society as a whole.[382]

---

[380]This assertion assumes that both licenses, the Apache license and the GPL, have full legal efficacy for their respective aspirations. The gap, however, between the two licenses is so significant, that even if the most far-reaching provisions of the GPL, such as the extension provision, are of uncertain efficacy, the GPL still provides a comparatively greater measure of protection against privatization of the source code. *See supra* notes 104–44, 350–54 and accompanying text (comparing two licenses).

[381]The cost to use open-source software is arguably lower, but not zero. It is not zero because most open-source licenses allow for fee-based distribution. In addition, software ownership and computing resources are often measured in institutional environments by a "total cost of ownership" ("TCO") model. Thus, while the lack of royalties for open-source software may lower its total cost of ownership, it will not drive it to zero. Indeed, one could argue that the TCO for open-source software may be higher than traditional software because open-source products are typically geared toward sophisticated users and may not be as "user-friendly" as traditional software. *See* Evans, *Preferring OSS*, *supra* note 129, at 40, 42 (discussing TCO model, "which includes training and support," and open-source software's failure to make "inroads in most business—and household—software categories").

[382]Each regime claims beneficial features for society, but opposing contentions dispute the purported benefits. For moral rights, the claimed benefits of protecting artists and preserving and promoting art are disputed by the contention that the preservation effect is blunted and the additional transaction costs arising from moral rights, in aggregate, do more harm than good for artists. *See* Landes, *supra* note 295, at 8–11 (discussing proposition that integrity rights are inefficient where costs exceed benefits). Similarly, for open-source software, while it boasts private production of a public good, contentions disputing the benefit include: that total cost of ownership is actually higher with open-source software, that government support of open-source software is anti-competitive, Evans, *Preferring OSS*, *supra* note 129, at 42–47, that open-source software may produce too many versions of the software too fast, and that the inability of an open-source project to charge for and capture rents is counterproductive because it leaves the software underdeveloped for certain classes of uses and users. While the arguments on both sides for both regimes have some force, the benefits of open-source software seem more verifiable by simply gauging the growth in use of open-source software generally, and particularly among the flagship products such as Linux or Apache.

5. Effects on the Author, Artist, or Open-Source Programmer

Similar to the analysis for external effects, equivalency between open-source and moral rights should also show parallels in how violations of rights in both regimes affect the respective individual creators. Both types of works express personality, so violations in each case affront the personality of the creator. One classic argument explaining moral rights is that if the work embodies an extension or expression of the creator's personality, then modifying or mutilating the work harms the creator's personality. It is, in an imprecise analogy, akin to a metaphysical personal insult.[383] Understandably, an author or artist will be upset by the mutilation of her work. The event may rankle the work's creator. Even if she obtains a remedy enforcing the right of integrity, her enthusiasm for the work, or for a greater body of related work, may be dampened. Systemically, violations may produce incentives to create less risky works, curtail creative striving to some degree, use or make less controversial subject matter, or avoid certain venues or forms.

An open-source programmer may suffer similar effects if the key open-source license conditions are violated, resulting in privatization of the software, or in loss of attribution for the programmer. One model of the open-source movement posits a gift culture where collaborating programmers and users value gifts and the satisfaction that comes from giving.[384] A violation of the open-source conditions (in particular, the source code, no royalties, and reapplication provisions) disables the given source code gift. Unchecked violations create an instance, stream, or fork of the software that does not express the original gift intentions of the programmer when she contributed the software. Similar to the posited effects for right of integrity violations, open-source license violations can create counterproductive incentives for the individual programmer's future participation and, consequently, for the open-source movement as a whole. The anti-privatization power of the open-source approach is a foundation condition for open-source collaboration. When a programmer's source code is taken and privatized in violation of the open-source license, she will likely feel as if something was wrongly taken from her. Even though she intended to give away the source code, her gift was conditional, and it is reasonable for her to expect the conditions to be followed. Further, she may be less likely to contribute in the future after having suffered

---

[383]*See* Liemer, *supra* note 17, at 43.

[384]Raymond, *supra* note 189, at "The Hacker Milieu as Gift Culture" ("In gift cultures, social status is determined not by what you control but by what you give away.").

a violation, or may participate at a lower level of investment.[385] An example of the latter would be to desist in writing code but continue to use the software and dutifully report software errors and bugs when discovered. These reports are less directly subject to privatization and require less energy than generating new code.

As with the posited parallels for external effects, the parallels in this discussion are not fully identical. The creative medium in open-source software is more functional than the creative medium for traditional literary and artistic works. As a result, the metaphysical affront from violations in each area will have a different character. The painting of the statue may make hideous what was once beautiful. The unavailability of source code, or charging of royalties, cuts off collaborative opportunities for the source code, limiting its potential reach and distribution as open-source software.[386] Loss of beauty and loss of collaborative opportunity may be equally upsetting, but they are different losses.

Losses felt by the author, artist, or programmer reveal the metaphor between the open-source approach and moral rights. The link is through a model of personality. Affronts to the work are metaphysical affronts to the personality expressed in the work. Traditional literary and artistic subject matter, as well as open-source software, carry an expression of the creator's personality in different but analogous value-laden ways. This carried expression has external effects related to the expressed personality. In part, these external effects establish and infuse the creator's personality-based reputation. The aggregate implications of these phenomena, and in particular

---

[385]This portion of my argument depends on the assumption that (or takes its strongest form when) the programmer volunteered the code and was not paid to develop it. Estimates range on the question, but a non-trivial amount of open-source software is developed by programmers working in research settings or even in for-profit institutions such as Red Hat or IBM. These programmers might be expected to suffer less personal "harm" if the open-source conditions are violated for their code. While admitting that open-source code developed in this way weakens this point of the argument, it does not eliminate it, because not all open source is developed institutionally, and anecdotally, it is recognized that a sense of mission influences programmers. Although a programmer is being paid to work on an open-source project, the programmer's energy, productivity, and personal investment in the task is greater if a programmer buys into the movement.

[386]To say that violating the open-source conditions limits the software's reach as open-source software is not to say that the open-source approach is the best approach for a particular type of software. In some cases, privatizing the software or developing it as a traditionally licensed software product might extend its reach and market share beyond that attainable using the open-source approach. The question is how to determine which approach, open-source or traditional, is optimal for a particular application. *See* Raymond, *supra* note 125, at Part 10.2 (arguing that criteria to choose between open-source or traditional approach includes that one "can expect that open-source has a high payoff where (a) reliability/stability/scalability are critical, and (b) correctness of design and implementation is not readily verified by means other than independent peer review. (The second criterion is met in practice by most non-trivial programs.)").

their analogous reemergence in open-source software, show that a reputation and personality perspective of the open-source approach should not be lightly dismissed. Indeed, these comparisons teach the importance of these aspects to the open-source approach.

B. Right of Integrity Enforcement of the Open-Source Approach

The license conditions implementing the open-source approach, like the right of integrity, control the view that a work presents. This raises several implications, the first of which, discussed in the previous Section, is that each type of work is capable of expressing the personality of the work's creator. The second implication is the subject of this Section: the possibility that jurisdictions providing a right of integrity in software also may, through this right, give an additional basis for programmers to enforce the open-source conditions. This is not to say that on balance a right of integrity for software is a desirable thing. Indeed, such a right creates several potential problems for the open-source approach. Foremost, is the possible use of a software right of integrity to effectively bypass the open-source license conditions by attempting to govern modifications to the source code when the conditions expressly allow such modifications.[387] Despite these problems for open-source software, and other general concerns about moral rights in software, some jurisdictions provide these rights in software to some degree. By doing this, these jurisdictions illustrate another parallel between the right of integrity and the open-source approach.

In jurisdictions that provide a right of integrity in software, my second claim is that, under the perspective I put forth below, an open-source programmer may be able to use that right to enforce the key open-source license conditions. She would argue that such a license violation is a distortion, mutilation, or other modification detrimental to her honor or reputation.[388] This is an unorthodox use of the right of integrity, because it involves using the right to protect the opportunity for others to modify the work rather than prohibiting modifications. But at another level, it is similar: use of the right to prohibit a few very important meta-modifications. In other words, the programmer asserts the right of integrity to ensure that there are no

---

[387]*See supra* notes 335–37 and accompanying text (briefly describing how assertion of right of integrity in open-source software might look). A full discussion of all the potential ramifications of moral rights on the open-source approach, or on software development in general, is beyond the scope of this Article. My main point is that when provided by a jurisdiction, the right of integrity in software adds another layer of analysis to the dispute, aspects of which can cut both for and against the open-source approach.

[388]I have described the programmer's argument in the language of the Berne Convention's specification of the right of integrity. However, the actual claim would depend on the jurisdiction's codification of the right of integrity, and would be cast in that language.

modifications to the key nonnegotiable open-source license terms: source code availability, no royalties, and reapplication of the same terms to redistributions.

The text of the open-source license would be the centerpiece of the argument that an open-source license violation is a right of integrity violation.[389] Building on the parallels drawn in the preceding Section, the argument is that the open-source license expresses the integrity that should endue in the software. The software can only accrue to the programmer's honor and reputation if the source code is available, signaling the programmer's values in contributing to an open-source project and signaling her reputation in marshalling and invoking computing resources via the programming language. Indeed, if the software is redistributed without source code, it may harm the programmer's reputation, because the licensees of the redistributor will be unable to make changes to suit their unique needs. Moreover, within the open-source community, when the software is known to be open-source, having one's software in circulation without the source code can have reputational impact.[390]

The license terms support a right of integrity claim, because they are evidence that the expressed personality in the work is to partially dedicate the software to the public domain under the open-source approach. This is more than a statement of intention by the programmer, because she expects that the conditions will be followed.[391] If not, she suffers a personal injury when her source code gift is co-opted, and the beneficial external effects of the open-source approach are denied to a wider community.[392] It derogates her honor and reputation like painting the sculptor's statue.

---

[389]In critiquing my second claim, Mark Lemley offered the following intriguing perspective: that the GPL acts to "cut off authors' integrity rights by irrevocably pre-licensing derivative works." This perspective runs against my second claim, at least to the extent that it limits my second claim's attempt to associate the open-source licensing approach with the traditional right of integrity. Admittedly, my second claim depends on a recharacterization of the GPL's intent in-line with the purposes of integrity's anti-modification prohibitions. Such recharacterization, to the extent it works, probably does so only for the key aspects of the open-source approach: source code availability and no royalties for use.

[390]Reputational impact from open-source software circulated without the source code could take several forms. First, it could be taken as an omission that produces an inconvenience for the user. Second, it could shake the user's understanding that the software was open-source, perhaps raising questions as to whether it had been privatized. Finally, it may raise questions about the effectiveness of the project leader(s) in coordinating and facilitating the collaborative effort of the group.

[391]Regardless of any questions as to the legal efficacy of the open-source approach or any particular open-source licenses, by and large most licenses are followed because doing so beneficially establishes the collaborative foundation.

[392]The extent of injury felt by the programmer may be of lesser degree, or even non-existent, if she wrote the software at the direction of an institution that employs programmers to generate software that it releases into the open-source movement. *See supra* note 385 (highlighting effects of institutionally sponsored open-source programming).

There are limits to this proposal—not every license term violation would validly represent a loss of integrity in the open-source approach, but those that do are easy to assess. For example, failure to implement the notice provisions associated with disclaiming warranties and liabilities is an important, but peripheral, license provision not central to the integrity of the open-source approach. However, evaluation of the central open-source license provisions is straightforward, avoiding some of the difficult assessments necessary for the right of integrity. Judges need not aesthetically assess the work when deciding whether it has been mutilated or modified. There is no such awkward step for judges applying the right of integrity to the open-source approach, because it is straightforward to evaluate whether source code was made available. Similarly, judges can readily evaluate other aspects of the open-source approach. Determining whether royalties were improperly charged is similar to a court making an accounting assessment. Whether the open-source license terms were reapplied on redistribution is also a straightforward assessment.

If integrity in the open-source approach is a viable alternative basis to enforce the approach, this raises the question: when would the alternative be necessary or desirable? Since the open-source approach is based on conditional permission to use a copyrighted work, the necessity would arise whenever a programmer or group of programmers cannot enforce the copyright rights. This could occur for a number of reasons. Most of these, however, are avoidable reasons. The most likely possibility is that the programmer assigned the copyright rights to another person or entity, and the assignee is unwilling or unable to act. Or perhaps the programmer did not secure a reciprocal promise from the assignee to enforce the open-source license.[393] In contrast to situations where integrity in the open-source approach is the only alternative, because the copyright cannot be enforced, bringing a right of integrity claim alongside a copyright infringement claim is useful to increase the chances of forcing compliance with the open-source approach. By claiming in the alternative, the programmer increases the defense's burden and benefits her position in the enforcement action.

Besides its potential value as an alternative claim in an open-source enforcement action, this proposal suggests that courts could look to right of integrity cases as persuasive authority for the importance of upholding the

---

[393]There are several other less-likely possibilities as to why the programmer might be unable to enforce her copyright. The copyright may have expired and the software is still in use (and the right of integrity is still available). The programmer may not have registered the copyright in a jurisdiction requiring registration to bring suit and there is some obstacle to registering the copyright at the time of suit. A related issue is the question of beneficial ownership and the possibility to enforce the copyright even if one has assigned her copyright rights. *See* Kwall, *supra* note 25, at 47–51 (discussing possibilities of beneficial ownership under United States copyright law to enable copyright holder to assert rights even after assignment of copyright); *see also* NIMMER & NIMMER, *supra* note 55, § 12.04[B]–[C] (discussing United States copyright infringement standing doctrine and bringing suit as beneficial owner).

open-source approach as a copyright license. There are no open-source software cases available as precedent. Thus, for jurisdictions that provide a software right of integrity, bringing the alternative claim is one way to focus the court's attention on the parallels between moral rights and the open-source approach. Traditional software licensing and copyright infringement cases would be the main staple of precedent in a United States copyright infringement action over an open-source license. There could be value in supplementing this primary source of law with the perspective recorded in right of integrity cases, whether they be from the United States or international jurisdictions. The parallels are sufficiently strong that such an approach might help the court see the open-source approach from a different perspective—one with a credible and long international history and one with a presence in the United States for visual arts.

Although the United States does not provide a right of integrity in most copyrightable subject matter, some jurisdictions provide the right across the board, including a full-fledged right of integrity in software.[394] This creates both potential problems and opportunities for the open-source approach. The opportunity may have practical utility, allowing one to bring an alternative claim to enforce the open-source license based on integrity in the open-source approach. Beyond that use, this possibility further illustrates the parallels between the two regimes. This Section and the prior Section reviewed these parallels in two ways, first by showing how both regimes express the personality of their respective creators, and second, by suggesting that the right of integrity may apply directly to the open-source approach. The next Section combines and extends these first two insights for my third claim: that a specific right of Collaborative Integrity should be considered for open-source software.

C. The Potential for Collaborative Integrity

Having reviewed the parallels between moral rights and the open-source approach, where each system controls the view that a work presents and allows the respective creators to express their personality in their works, the third implication raised by these parallels is whether the open-source regime could

[394] *See supra* note 331 and accompanying text (touching on jurisdictional differences in moral rights provisions).

be improved by incorporating elements of the moral rights system.[395] Specifically, I argue that the legal community should evaluate an altered approach to protecting open-source software and I sketch an alternative model I call "Collaborative Integrity" for open-source software.

Taking a cue from the French dualist model of moral rights, as well as from VARA in the United States, where authors' rights are separate from pecuniary copyright,[396] in United States law, Collaborative Integrity could be implemented under a statutory public law mechanism as a right separate from the Copyright Act's current section 106 rights.[397] Collaborative Integrity would recognize the functional nature of software. As with the copyright-licensing-based open-source approach, it would provide a foundation for collaborative endeavor among programmers to share freely usable and modifiable source code. As a statutorily implemented right, it would bring greater certainty to the open-source software movement.

Evaluating Collaborative Integrity is important for a number of reasons. First, due to the often-raised questions about open-source licenses and their degree of enforceability, it makes sense to evaluate potential alternatives. Second, the open-source approach is growing internationally, in some places

---

[395]The parallels that exist between open-source software and the right of integrity are not exclusive to this pair. Other technology licensing approaches also offer a rough parallel to the right of integrity, perhaps even to a greater degree than open-source software. In this vein, Mark Lemley suggested the example of Java, a programming language standard and technology that Sun Microsystems licenses to other vendors, requiring the venders to conform to the standard—which analogizes to keeping a sense of integrity about the standard for interoperability. *See* David McGowan, *Has Java Changed Anything? The Sound and Fury of Innovation Litigation*, 87 MINN. L. REV. 2039, 2041–49 (2003) (discussing aspects of government's antitrust action against Microsoft as it relates to allegations that Microsoft implemented nonconforming version of Java standard). Indeed, when Microsoft implemented a nonconforming version of Java, Sun was able to bring suit under the license to encourage conformity. Sun Microsystems, Inc. v. Microsoft Corp., 188 F.3d 1115, 1117 (9th Cir. 1999).

[396]17 U.S.C. § 106A(a) (VARA's moral rights are "independent of the exclusive rights provided in section 106").

[397]*See* Madison, *supra* note 245, at 338 n.221 (in a critique of contemporary software licensing, noting the potential for moral-rights-like rights for licensing open-source software); *see also* Yochai Benkler, *Through the Looking Glass: Alice and the Constitutional Foundations of the Public Domain*, 66-SPG LAW & CONTEMP. PROBS. 173, 194–95 (2003) (noting that open-source licenses implement rights securing the programmer's attribution and integrity in the software).

more rapidly in certain ways than in the United States.[398] One has less confidence in a copyright-based licensing approach when the licenses may need enforcement in separate international copyright jurisdictions, many of which have at the heart of the open-source approach nuances in the copyright infringement and licensing law. Third, the similarities highlighted in the previous two sections suggest that it is important to sketch Collaborative Integrity in order to fully explore the learning possible from comparing the two systems.

I proceed in two phases. First, I discuss the reasons supporting a separate right of Collaborative Integrity and some of the possible objections. Collaborative Integrity should further heighten the incentives for the creation and contribution of open-source software. At a policy level, it would express an approval of the volunteerism inherent in the open-source movement, which has produced valuable public goods, in part due to substantial private effort. It could facilitate evolving forms of open-source project organization and management. Alternatively, it could supplement the enforcement power of open-source licenses that seek to attach additional conditions on open-source use. Second, I sketch the rough contours of the right of Collaborative Integrity. It should certainly include the following central elements from the open-source approach: source code availability, royalty-free use, rights to modify and redistribute, and reapplication of these same terms to redistributions. These elements provide the collaborative foundation. However, to further specify the right requires addressing whether it should include the extension provision—that feature of the GPL that sought to extend the GPL's terms to software coupled to the GPL-licensed software in particular ways. In addition, questions linger about how long the right should persist, and whether it should be waivable or assignable.

1. Rationale for Collaborative Integrity

Given the doctrinal uncertainty about the open-source licensing system, Collaborative Integrity should heighten programmers' incentives to contribute source code by making the rights more certain. Programmers, relying on the

---

[398] *See* Evans, *Preferring OSS*, *supra* note 129, at 34 & n.1 (describing legion of countries considering proposals to make open-source software official preference). Government preferences for open-source software could include preferences in its use, that is, choosing open-source software due to perceived cost savings and customization opportunities. *See* Byron Acohido, *Linux took on Microsoft, and won big in Munich: Victory could be a huge step in climb by up-and-comer*, USA TODAY, July 14, 2003, at B1, *available at* 2003 WL 5315241 (describing how the city of Munich, Germany, decided to use Linux rather than Microsoft's Windows operating system). Government preferences could also include contributing source code to open-source projects. Governments at all levels write, or have written, substantial amounts of code, raising the possibility that if they contributed the code to open-source projects the code would bring greater value to a wider group. Lessig, *supra* note 89, at 65.

expectation that their source code will remain open, are motivated to contribute. Developers on large open-source projects experience the various satisfactions from open-source development only if the open-source approach is respected and provides a foundation for collaboration. For example, a programmer receives much less reputational impact if the software is promulgated without source code.[399] Thus, a legal regime providing greater certainty for the open-source approach increases the programmers' estimation that the desired outcome will occur, assuming she is aware of the legal dimension of her choice. There is evidence that programmers who start open-source projects are aware that they should choose a license for the project,[400] signaling an awareness of the importance of the legal rights underpinning the project. Greater certainty in the rights that keep source code available should increase the incentives to contribute, or at least decrease potential hesitancy to contribute.

The ex-ante incentives deriving from more certain rights under Collaborative Integrity are in contrast to, but supported by, ex-post approval of a programmer's contribution, and the open-source approach in general, that Collaborative Integrity would express. Although copyright-based private-law generally-applicable open-source licenses implement the open-source approach, they do not embody a societal judgment endorsing open-source. As a statutory regime, Collaborative Integrity would express this approval. It would provide the open-source movement with express endorsement of the approach and the volunteerism inherent in the community.[401] One could argue that copyright law already expresses the appropriate policy approval, because one of its goals is to generate the production of creative works that will eventually fall into the public domain for all to use. Open-source accomplishes a very

---

[399]That a programmer obtains less reputational impact (as opposed to no impact) assumes, as is typically the case, that the software contains authorial identifiers external to the source code. These are sometimes found in ancillary files in the distribution, or in the software's introductory screen(s) or "help" system. With effective attribution, reputational incentives can be an important force generating copyright subject matter. *See* Ryan, *supra* note 170, at 652 n.25 (noting that legal academic literature pays little attention to noneconomic reasons for creative production, and citing others for the proposition that "reputational rewards achieved through wide dissemination provide far more creative incentive than the financial reward . . . connected with the licensing and sale of individual copies of work'" (citations omitted)).

[400]Lerner & Tirole, *Scope of Licensing*, *supra* note 98, at 1–2, 8–10 (describing survey of approximately 40,000 open-source projects housed at SourceForge.com open-source repository, survey intending to explore factors influencing open-source developer's choice of license for project).

[401]*See* Landes, *supra* note 295, at 19 (noting that state moral rights laws may have produced better social environment for artists in those states). If state moral rights laws can produce a better environment for artists in a state, then Collaborative Integrity can also produce a more conducive environment for open-source programmers. On the other hand, however, one might object on the basis that there are other ways to foster this environment, such as by government preferences in using and contributing to open-source software. *See supra* note 398 (citing success of such approach in Munich, Germany).

similar goal, perhaps more effectively than copyright, and certainly more quickly. But just because open-source happens to support a long-term copyright goal does not mean that copyright policy fits open-source. The inversion between the systems is too great because the methods are so different. Open-source software is such a unique reversal of the copyright paradigm that it deserves its own protective right, Collaborative Integrity, and the attendant social and policy approval inherent in vesting the right.[402]

Collaborative Integrity, as a separate right from copyright, could facilitate new arrangements for open-source project repositories, organization, and governance. It would allow a programmer to assign her copyright in the software to another person or entity, yet still retain her right of Collaborative Integrity. Some open-source project repositories and institutions already request such assignments from contributing programmers.[403] And some commentators have suggested the desirability of a public trust or conservancy approach to house open-source software.[404] The potential desirability of these approaches stems in part from the unavoidable baggage that comes from copyright in software and the license as a species of contract. The problems fall in two main areas: infringement related problems and other problems.

The recent lawsuit by SCO against IBM provides an example of infringement related problems. SCO is a small company which allegedly holds rights to portions of certain Unix operating-system software, and which, either directly or via its predecessors in interest, had licensed the software to a wide variety of vendors. SCO sued IBM for alleged violations of SCO's software license agreement with IBM. In the agreement, IBM promised not to disclose trade secrets in Unix source code that SCO provided to IBM. SCO alleges that IBM copied some of this code into Linux, thus disclosing the trade secrets, because Linux is open-source software. If SCO's allegations are true, it has a copyright infringement case against the many users of Linux distributions that contain this IBM-supplied code.[405]

---

[402]*See* Goldstein, *supra* note 302, at 1120–22 (arguing that copyright protection is workable solution to protect software, but acknowledging that other options might include "a new intellectual property system specifically tailored to attract investment toward the desired level and objects of innovation").

[403]Eben Moglen, *Why the FSF Gets Copyright Assignments from Contributors*, *at* http://www.gnu.org/licenses/why-assign.html (updated Jan. 29, 2004); OpenOffice.org, *Joint Copyright Assignment Form*, *at* http://www.openoffice.org/licenses/jca.pdf (updated Sep. 10, 2002) (implementing joint assignment of copyright in contributed code).

[404]*See* Benkler, *Coase's Penguin*, *supra* note 4, at 446. *See also* Ryan, *supra* note 170, at 705–06 (proposing public trust model for copyright and information works).

[405]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 supra* note 336 (noting disruptive effect and potential liability of Linux users stemming from SCO suit).

Among the many potential ramifications of this lawsuit is its illustration of a competitive disadvantage for open-source software in the eyes of corporate and enterprise customers: lack of intellectual property indemnification. The major Linux distributors, for the most part, do not provide this indemnification, because the source code in a Linux distribution comes from thousands of programmers who hold an interlocking web of copyright ownership in the software.[406] This disperse ownership group, while overcoming collective action problems to develop and combine their software, has not come together to establish a system that would allow indemnification. Enterprise customers would not accept indemnification from such a disperse group anyway; the indemnification would need to come from a major market participant such as IBM or Red Hat. Indeed, one vendor, Hewlett Packard, has offered a limited indemnification for Linux running on its hardware.[407] The most effective way for a central entity to provide indemnification is to own all

[406]Charles Cooper, *The Next Big Linux Controversy*, CNET News.com (July 18, 2003), *at* http://news.com.com/2010-1071_026988.html (last visited July 20, 2003) (noting that, as of date of news report, none of Linux distributors or resellers, including IBM and Red Hat, offers intellectual property indemnification to their licensees). Microsoft, on the other hand, has recently extended and broadened its intellectual property indemnification provisions to stress this comparative advantage over Linux. *See* Michael Kanellos, *Microsoft Easing Customers' Legal Stress*, CNET News.com (July 22, 2003), *at* http://news.com.com/2100-1012-5050986.html (last visited Feb. 12, 2004) (noting that "Microsoft has a new sales pitch for Linux users: Buy our software and stay out of court."). More recently, however, one vendor, Hewlett Packard ("HP"), in response to user concerns about the SCO suit, began offering an indemnification for Linux users who ran the operating system on HP's hardware and met other conditions. *See* HP Linux Indemnity Site, *at* http://.www.hp.com/wwsolutions/linux/linuxprotection.html (last visited January 9, 2004) [hereinafter HP Indemnity] (describing HP's indemnity program, including requirement that it applies only to Linux purchased from HP or its authorized resellers, which runs on HP hardware, and which customer has not modified).

[407]*Id.* at 406.

the copyright rights.[408] Even then, to be confident in granting the indemnification, the central entity would need procedures to ensure that contributed code is from a true author and does not carry a copyright infringement risk.[409] While difficult, these screening procedures could be implemented with some degree of success. If implemented at the beginning of a project, a trusted central entity could own all the copyrights in the collaborative project with sufficient confidence to take the risk of granting indemnification to enterprise customers.

This brings us back to the role of Collaborative Integrity—which is to eliminate the central entity's need to rely completely on programmer's entrustment. With the right of Collaborative Integrity, open-source programmers could confidently assign their copyright ownership to the central entity, knowing that they could enforce the open-source nature of the software with Collaborative Integrity. For copyright related issues, the central entity then has no need to overcome collective action problems, because it owns all the copyrights in the project, putting it in a better position to deal with infringement issues. Besides providing indemnification, the central entity would be better positioned to participate in the legal process if the open-source software is accused of infringing someone else's intellectual property, be it

---

[408]In asserting that the best approach is for the central entity to take copyright assignments, I acknowledge that there are a number of second best alternatives to outright ownership. In this discussion, I am assuming that the central entity is also the aggregator and development leader for the open-source software, or at least is coordinating and cooperating with the development leaders. First, the entity could obtain indemnification from the individual contributors, and on that basis offer an indemnification to enterprise customers. This, however, gives the central entity a frail basis for its indemnification, because it is unlikely to take recourse against the disperse development community. The entity will not want to alienate them, and their assets may be insufficient. Second, the entity could have contributors grant licenses to the entity. From these rights, the entity would issue the generally applicable open-source license. This basis is also frail, because as long as the contributors are record owners of the copyright, they could grant conflicting licenses. Putting aside the problem of the author's reversion right in United States copyright law, 17 U.S.C. § 203(a)(3) (2002) (enabling certain authors to terminate grant of a copyright or rights under a copyright "at any time during a period of five years beginning at the end of thirty-five years from the date of execution of the grant"), record ownership by the central entity eliminates conflicting rights arising from the contributing developer after she contributes the code, assuming that the developer is the true author and did not violate anyone else's copyright rights in contributing the code.

[409]The central entity has an important incentive to ensure that programmers assign the copyright: blocking submittal access to the open-source repository housing the project until the assignment is completed, submitted, reviewed and approved. *See* OpenOffice.org, Contributing to OpenOffice.org, *at* http://www.openoffice.org/contributing.html (updated Sep. 10, 2003) (noting that programmer must sign copyright assignment if their code is to be integrated into open-source project). Beyond these formal mechanisms, one can imagine that a programmer who submits code copied from others, thus endangering the open-source project by embedding a future copyright infringement risk, if discovered, would be subject to sanctions from the community directly—such as effectively blackballing the programmer from future project participation.

copyrighted material or a patent.[410] While programmers might be willing to trust a nonprofit central entity with their copyright assignment, they probably would have less trust in a for-profit entity. Collaborative Integrity would nullify that concern, allowing a broader array of open-source organization and management structures.[411] Many in the open-source community view greater involvement by for-profit entities as important to the growth of open-source software.[412] Such entities are better positioned to reach certain markets, and there are a number of open-source business models that have proved viable, enabling entities to achieve commercial success without the traditional closed source royalties-for-use software model. By facilitating centralized copyright ownership and other new structures, Collaborative Integrity supports new arrangements that help deal with infringement issues and other issues facing open-source software. Foremost among the other issues facing open-source software are warranties and liabilities. If a central entity owns all the copyrights in the project, it is in a better position to grant some degree of

---

[410]On the flip side of this issue, although open-source software projects are probably less likely to be litigious than traditional business concerns, centralizing copyright ownership would simplify issues of standing for enforcing the copyright and open-source licenses based on the copyright. In addition, this approach may particularly benefit the leaders of an open-source project. Assume the central entity owns the copyright, rather than the individual project leaders themselves. In the case of an infringement lawsuit against the open-source software, the leaders may benefit to some degree from the interposed central entity, as they will not be personally liable.

[411]If the primary issue were simply trust between the programmer and the central entity, an alternative way to approach the problem is by contract between the for-profit central entity and the programmer, where the central entity agrees to use copyright to uphold the open-source nature of the software. While such an approach might approximate the beneficial effect of Collaborative Integrity, it also carries with it the baggage that comes from copyright in software and the license as a species of contract: potential lack of uniformity within the United States under state contract law; the effect of such contracts in the cases of changes to the entity, such as bankruptcy, merger or acquisition; classification issues for software-related transactions under state contract law; and the potential for federal copyright preemption to vitiate license terms. *See generally* Madison, *supra* note 245, at 276 (noting various issues that affect legitimacy of licensing regime).

[412]Raymond, *supra* note 125, § 3 (discussing why open-source software "increasingly poses . . . an economic challenge" and noting need for price structure "[u]nder the efficiency seeking conditions of the free market").

warranty or liability coverage, because these typical licensing terms are often demanded in the same class of situations where indemnification is important.[413]

Enforcement of the open-source approach is another issue. Collaborative Integrity, in the hands of the programmers, acts as a supplemental enforcement mechanism, backing up the central entity's efforts. It constrains the central entity, who will still license the software under a copyright-based open-source license, by requiring its license to conform with Collaborative Integrity. It similarly constrains users from violating Collaborative Integrity. As a disperse group, the programmers might be thought to be ineffective at enforcement.[414] But, being decentralized, they are also less subject to a failure of agency that might result in the central entity privatizing the software or deviating from the open-source approach.[415] As the open-source movement grows, projects are experimenting with new licenses and new forms to organize both legal rights

[413]Since warranties and liabilities sound in state law, an alternative approach to this issue is specific state law exemptions for any implied warranties and liabilities when free or open-source software is at issue. This approach has been implemented to a limited extent in relation to UCITA. *See supra* note 257. If the open-source software is truly free, that is, there is no distribution fee whatsoever, then this approach seems palatable, although, assuming that not every state would enact such exemptions, it may result in patchwork coverage. But distribution fees for enterprise versions of Linux can run into the four-figure range, *see supra* note 156 (citing Linux prices), sufficiently high to raise questions about the equitableness of exempting all warranties and liabilities.

[414]This question also arises for moral rights' role in preserving cultural and artistic heritage for literary and artistic works. *See* Cotter, *supra* note 17, at 74 (discussing how "society that endows artists with moral rights necessarily incurs costs related to enforcement . . . of those rights" and these costs, in part, lead author to conclusion that perhaps moral rights cannot "be justified an economic grounds as a means of correcting for a failure in the market for cultural preservation"); *see also* Hansmann & Santilli, *supra* note 17, at 106 (noting existence of "European statutes that protect and preserve those works that are considered important to nation's artistic heritage").

[415]Agency failure by the central entity holding all copyrights could occur in varying degrees of severity. If it tried to privatize the entire project, assuming no explicit contractual promises to the developers to enforce the open-source approach, the contributing programmers might be left with only pleas to equity. Less severe might be a central agency's attempt to write an anti-forking license—one that only allows modifications if they are sent back to the central entity. *See, e.g.*, Bruce Perens, *The Open Source Definition*, *in* OPENSOURCES, *supra* note 2, at 184 (describing how when Netscape released source code for its Web browser, which resulted in Mozilla open-source project, Netscape implemented license (Netscape Public License ("NPL")) that gave only Netscape right to make submitted modifications private—by reincorporating them back into its propriety browser). The power to fork the project is important to project functioning because it gives power to the voice of the distributed programming group if they disagree with the project leaders. *See* Raymond, *supra* note 125, § 8. Netscape has since moved away from the NPL to a new licensing scheme. Mozilla Relicensing FAQ, *at* http://www.mozilla.org/MPL/relicensing-faq.html (last modified Dec. 7, 2003).

and collaborative development.[416] Mechanisms that support such experimentation should be encouraged. Collaborative Integrity would contribute to the experimentation because it is partitioned from copyright, an approach suggested by the dualist moral rights implementations.

Reviewing the possible rationales for Collaborative Integrity illustrates the insights from comparing moral rights to open-source software. Collaborative Integrity would heighten incentives for open-source programmers, express policy approval of their contributive activity, and make possible new ways to organize collaborative effort, yet preserve an enforcement right with the original contributors. The next step to assess Collaborative Integrity is to estimate its contours, or at least assess the factors that would determine a full-fledged right of Collaborative Integrity.

2. The Contours of Collaborative Integrity

Exploring the feasibility or desirability of Collaborative Integrity for open-source software requires evaluating choices as to the content of the right. Under one specification the right may be desirable; under another it may be undesirable. I touch upon these choices in two sets. First, those features that support collaborative software development and form the core of the right. Second, the ancillary features necessary to define the right's operation. I do not completely develop a formal analysis for the second set, but suggest what, in my view, are the most likely features. The choices for the first set are more apparent, deriving from the core goal of the right—supporting collaborative open-source software development.

The core of Collaborative Integrity should reconstitute the primary open-source license conditions that enable open-source developers to create software: source code availability, royalty-free use, modification and redistribution, and reapplication of at least these same terms to redistributions. These rights give the code the transparency that comes from source code availability, and the ease of sharing from royalty-free rights to use, redistribute, and modify. These rights should persist and "run" with the code—suggesting a feature similar to the reapplication provision found in some open-source licenses. How long they should run with the code is a question for the second set of features. The most troubling issue for the core specification of Collaborative Integrity is whether it should include the extension provision, and, on a related note, how to effectively differentiate that provision from mere modifications to the open-source software.

---

[416] *See* Mozilla Relicensing FAQ, *supra* note 415 (licensing source files under "triple license" that includes GPL and LGPL); *see also* OpenOffice.org, Licenses, *at* http://www.openoffice.org/license.html (updated Aug. 12, 2003) (describing OpenOffice project's use of dual licensing strategy).

The extension provision is the feature of the GPL that seeks to extend the GPL's terms to software coupled to the GPL-licensed software in particular ways.[417] The context differentiates it from simply modifying and evolving the open-source software. The extension provision envisions other, non-open-source, previously existing software that becomes intimately coupled with the open-source software. The fact that the two separate sets of source code are operatively coupled (perhaps at the object code level), rather than indistinguishably intermingled at the source code level, implies a continued partition and technological separateness. Given this partitioning, the core protective function of Collaborative Integrity should not extend to the non-open-source software. That software was not developed in the open-source tradition and does not carry the moral-rights-like attributes suggested by my comparative analysis.[418]

The second set of choices for the contours of Collaborative Integrity primarily concern the duration of the right, whether it should be waivable or assignable, and the issue of remedy. The French right to respect implements a strong form for these features. The right is perpetual, passing to the creator's heirs. It is neither assignable nor waivable.[419] VARA, in the United States, implements the opposite approach. The VARA right of integrity persists for the artist's life, and is waivable but not assignable. The VARA approach to these features would likely be sufficient for Collaborative Integrity for several reasons. First, most programmers outlive their code. The useful life of most software is short compared to the useful life of fine art. Thus, there is likely no reason for Collaborative Integrity to extend beyond the life of the programmer. On the other hand, as long as the programmer is alive, she has a personality expression interest in enforcing Collaborative Integrity, in particular to protect her reputation.

Further, whether the right is assignable and waivable influences the flexibility of the right. A waivable right facilitates both traditional software

---

[417]*See supra* text accompanying notes 209–217 (discussing GPL license and its attributes).

[418]In addition, the extension provision has been a lighting-rod for criticism of the open-source approach. Collaborative Integrity need not be antagonistic to non-open-source software because its primary goal is to facilitate and enable open-source as a competing, not supplementing, regime of software development. Commentators acknowledge that both styles of development have a place in the market and each fit with particular classes of applications. Raymond, *supra* note 125, §§ 2, 10 (noting that argument for open-source software does not rest on notion of closed source software as "wrong" and suggesting when software development should be open and when closed).

[419]Although seemingly quite powerful, even the French right to respect is tempered by doctrines that allow reasonable modifications in certain situations. *See* Kwall, *supra* note 25, at 13 (noting that "the French judiciary tends to enforce contracts allowing reasonable alterations that do not distort the spirit of the creator's work, particularly with respect to adaptations and contributions to collective works").

production and open-source production,[420] and makes sense on that basis. The programmer can waive the right if she makes the decision to work for a traditional software concern. An assignable right goes beyond waiver. Besides giving up her right to enforce Collaborative Integrity, the programmer transfers the enforcement power to someone else. Following VARA, and at least from a personality interest perspective, the right need not be assignable because the programmer is the person most interested in protecting her personality expression in her contributed open-source software.[421] Finally, although VARA defines violations of its right of integrity in such a way as to include the Copyright Act's infringement remedies, Collaborative Integrity seems to be best served with exclusively a property rule injunctive remedy.[422] Damages are too speculative for open-source software because there are no market transactions for software royalties to establish valuations. Disgorgement of the violator's profits from violating Collaborative Integrity might make sense, but there will often be intractable problems to disaggregate an allocation of such disgorged amounts among the contributing programmers on the project.[423]

Further analysis could produce a more detailed specification of the contours of Collaborative Integrity.[424] For this initial sketch, however, the

---

[420]Commentators generally acknowledge that both types of software production are likely to persist. *See, e.g.*, Bradford L. Smith, *The Future of Software: Enabling the Marketplace to Decide*, *in* GOVERNMENT POLICY TOWARD OPEN SOURCE SOFTWARE 69, 70 (Robert W. Hahn ed., AEI-Brookings Joint Center for Regulatory Studies (2002)), *at* http://aei-brookings.org/admin/pdffiles/phpJ6.pdf (last visited Jan. 8, 2004) (noting that "*both* open-source and commercial software are integral parts of the broader software ecosystem").

[421]At least, this is the case when the programmer volunteers her time to write and contribute the code. Some open-source programmers, however, are paid to write the software, and this posits the counterargument that perhaps the right should be assignable in those cases to the employing entity.

[422]An injunctive remedy would most likely operate against distributors of non-conforming software, that is, against software distributed without source, or for which the distributor demanded ongoing royalties for use. A more difficult question is whether the injunctive power should reach a user of such software, who might not have knowledge of the violations.

[423]Assumedly, in most cases an entire functioning project will be privatized, not just one programmer's code, unless the project is a sole programmer effort. Even with the difficulties of allocating any disgorged amounts to a programming team, it would seem necessary to disgorge, because leaving ill-gotten profits with a wrongly-privatizing person creates perverse incentives. Allocation of disgorged amounts could perhaps follow copyright doctrines of joint authorship and ownership as applied to an open-source project, but the applicability of the traditional doctrines may be awkward given the unique nature of open-source development. *See* Severine Dusollier, *Open Source and Copyleft: Authorship Reconsidered*, 26 COLUM.-VLA J.L. & ARTS 281, 292–94 (2003) (discussing authorship under open-source model).

[424]Other issues to be specified include whether there would be a notice requirement, i.e., would programmers have to specify in the software that it was covered by Collaborative Integrity. In addition, is the question whether, as in VARA, 17 U.S.C § 106A(b), one has to be an "author" under the Copyright Act in order to qualify for protection. Software is thought to only qualify for "thin" copyright protection. This raises the question whether coverage of Collaborative Integrity should be coterminous with coverage of copyright in computer program source code.

foregoing illustrates one set of rough dimensions for the right, intended to show that a workable right is possible. Besides the feasibility inquiry, several moral-rights-inspired reasons argue for Collaborative Integrity. These include heightening incentives for programmers in light of the nascent nature of the open-source approach, facilitating new institutional arrangements for the open-source movement, and expressing policy approval of the movement's approach, methodology, and benefits to society in producing public goods through private effort.

VI. CONCLUSION

Open-source software is a new order full of transposition and paradox. Copyright licenses promote copying rather than prohibit it. Software teams work better scattered around the world than housed in one place. Free software has higher quality than sold software. The open-source phenomenon is part ideological triumph and part opportunistic pragmatism. There are many facets to this new phenomenon, many ways to better understand what originated it, sustains it, helps it grow, and what may threaten it.

The movement finds its foundation in norms encoded in a nascent copyright-based licensing scheme, which sets the stage for new forms of collaborative, Internet-enabled, distributed software development. An impressive, even stunning, array of software has resulted so far. The norms stress source code availability, collaboration among project leaders, developers and users, and a continuing legacy that the source code remain freely useable, modifiable, and shareable. The norms emphasize freedom for programmers to continue to do these things—things which have been denied them by the status quo of traditional software development.

Open source is a modern phenomenon, but another movement with many parallels originated over two hundred years ago in France. The system of *droit moral*, or author's moral right, grew through French jurisprudence in response to that society's felt necessity to protect and value creative works and the authors and artists who produced them. In similar fashion, open-source software has emerged to protect and value software transparency and freedom. The resulting collaborative foundation has produced much of the infrastructure software for the Internet. Open source is, in part, a response to the felt necessities of our time. The parallels between the two systems are numerous and exist at several levels. At the heart of both systems is the right for creators to control the view that a work presents, to keep the work in a certain beneficial state. This is the right of integrity in the civil law moral rights tradition. In the open-source system, it is the Collaborative Integrity of open-source software.

Moral rights, with its history and legacy, can help us better understand the Collaborative Integrity in open-source software. Moral rights seek to protect

the creator's personality as embodied in the work. The open-source approach similarly protects the transparency necessary to show the programmer's personality in contributing to a project. The moral right of integrity, in some jurisdictions, may apply to software, thus raising questions whether it hurts or helps the current copyright-based licensing approach to protecting open-source software. From these two moral-rights-inspired insights, a third insight arises: does the Collaborative Integrity in open-source software deserve protection as a right of its own, just as the right of integrity developed separately from pecuniary copyright in some civil law jurisdictions? These insights, and the questions they raise, demonstrate how comparing the venerable moral rights tradition to the open-source approach helps us better understand many aspects of this new phenomenon and better understand the Collaborative Integrity in the open-source movement.