Maine Law Review

Volume 58 | Number 1 Article 8

January 2006

Rethinking UCITA: Lessons from the Open Source Movement

Matthew D. Stein
University of Maine School of Law

RETHINKING UCITA: LESSONS FROM THE OPEN SOURCE MOVEMENT

RETHINKING UCITA: LESSONS FROM THE OPEN SOURCE MOVEMENT

> Open source software is an idea whose time has finally come. For twenty years it has been building momentum in the technical cultures that built the Internet and the World Wide Web. Now it's breaking out into the commercial world, and that's changing all the rules. Are you ready?[1]

I. INTRODUCTION

For those within the information technology (IT) industry, the phrase "open source" has been as prominent at water cooler and boardroom discussions over the last several years as the phrase "out source." Open source is at once a software development model, a business model, a social movement, and a philosophy that has recently garnered attention from outside of the IT sphere. As such, the topic has become increasingly fertile ground for academic scholarship from several disciplines. Economists, legal academics and practitioners, computer engineers, and social commentators have offered their varying perspectives on open source software. Whether or not this attention is warranted, and whether or not this is truly "an idea whose time has finally come," remains unclear. In fact, clarity and certainty are not adjectives that fit the open source movement particularly well. Indeed, from a legal perspective, there are more questions than answers in this area at present. Despite these legal uncertainties, some open source software projects have flourished in the United States and abroad. Several open source software projects also serve as critical infrastructure for the World Wide Web.

Beyond open source software specifically, there are also numerous and important questions about the legitimacy and enforceability of mass-market software license agreements in general. Courts that have been asked to interpret these agreements have offered various approaches and conclusions. Some courts have applied Article 2 of the Uniform Commercial Code (UCC) to these transactions; others have held that it does not apply. Given the increasing importance of computer software in our economy and the widespread use of these license agreements, consistent and uniform rules would be desirable.

A recent effort to codify uniform rules to govern software licenses appears to be languishing. The National Conference of Commissioners on Uniform State Laws (NCCUSL) recently proposed the Uniform Computer Information Transactions Act (UCITA) for adoption by all the states. At present, only two states have adopted UCITA and the prevailing logic suggests that further adoption will be an uphill struggle. In large part, UCITA was controversial because it was perceived as overly protective of large commercial computer software developers–most notably, Microsoft. A diverse and energetic collection of interests, aligned against UCITA, has succeeded thus far in derailing its progress. Although the argument that Microsoft is hampered by a lack of uniformity and certainty in the law is not likely to engender a great deal of sympathy, that same argument in the context of open source software might be more

---

1. Open Source Initiative, http://www.opensource.org/ (last visited Sept. 20, 2005).

convincing. In this Comment, I argue that the open source movement necessitates a rethinking of UCITA, or at least a UCITA-like uniform code to govern software licensing transactions. If UCITA benefits the open source movement, then former opponents may be willing to take another look at the statute.

Part II of this Comment offers some basics of software programming and the relevance of those basics to the open source approach. Part III provides some background by introducing the legal protection for computer software under existing intellectual property laws. Part IV builds on this legal background by examining the evolution of the software licensing regime. Courts have failed to offer consistent guidance in this area of the law. Part V explores the trials and tribulations of UCITA, from its earliest stages to its current static existence. Part VI provides a general introduction to the open source movement, including a discussion of the various open source licenses that are currently being used. What started as a limited group of academic and hobbyist programmers sharing source code has become a prominent landmark on the software development landscape. Open source software now serves as the foundation of several successful businesses. A brief overview of the various open source licenses is also included in this section. Part VII includes a discussion of the legal uncertainties facing the open source movement and speculation on the extent to which these uncertainties might be limiting broader participation in the open source approach. Part VII also offers a brief analysis of the impact that UCITA might have with regard to the open source movement and a discussion of what lessons might be learned by thinking about UCITA and open source software simultaneously. For a number of reasons, a rethinking of UCITA, in light of the open source movement as well as some important amendments to UCITA itself, leads to the conclusion that UCITA should now be adopted by the states.

II. SOFTWARE PROGRAMMING 101: SOME BASICS

In order to better understand the legal issues raised by the development of open source software, it is appropriate to provide a brief introduction to the technology of software programming. The term "source" in "open source" refers to source code. Source code is the language that programmers use to both speak to computers and command them to execute desired functions.[2] Source code has been described as the "holy grail" of the open source movement.[3] Actually, it is technically incorrect to say that programmers speak directly to computers because computers do not directly read nor respond to source code. Programmers and computers do not speak the same language.[4] Computers respond to another type of language, usually referred to as "object code" or "executable code."[5] Object code is a binary language of consecutive ones and zeros that controls on/off switches in the computer hardware.[6] Programmers

---

2. *See* Mathias Strasser, *A New Paradigm in Intellectual Property Law? The Case Against Open Sources*, 2001 STAN. TECH. L. REV. 4, ¶¶ 8-9.

3. Greg R. Vetter, *The Collaborative Integrity of Open-Source Software*, 2004 UTAH L. REV. 563, 578.

4. *See* Daniel B. Ravicher, *Facilitating Collaborative Software Development: The Enforceability of Mass-Market Public Software Licenses*, 5 VA. J.L. & TECH. 11, ¶ 7 (2000).

5. *See* Strasser, *supra* note 2, at ¶ 12.

6. Shawn W. Potter, *Opening Up to Open Source*, 6 RICH. J.L. & TECH. 24, ¶ 7 (2000).

use a number of different programming languages to write source code, all of which are readable and understood by humans.[7] These programming languages are the expressive medium through which programmers dictate the desired function of a program. The source code is then translated into the computer-readable object code before the computer is able to respond to the commands of the programmer.[8] This process is called compilation.[9]

This distinction between source code and object code is an important one because it lies at the heart of what distinguishes traditional proprietary software from open source software.[10] The key difference between open source software and proprietary software lies in the development and distribution method. Traditionally, commercial software developers have utilized the proprietary model, distributing software in object code form only and keeping the source code secret.[11] Open source proponents have sharply criticized this model of software distribution on both moral and practical grounds, supporting free and open access to source code.[12] They liken the proprietary distribution model to selling a car with the hood welded shut.[13] A car buyer in this situation would have no way of knowing how the car operates and would be unable to repair any internal problems themselves. Although many, if not most, car owners care little about what is under the hood, never mind being able to make their own repairs, there are some that would at least like to have this option. This too is the case with

---

7. "Software programmers write software programs using a high level computer language such as BASIC, C++, or Java. These high level languages generally use English alphanumeric characters and symbols and enable the developer to tell the computer what to do." Christian H. Nadan, *Open Source Licensing: Virus or Virtue?*, 10 TEX. INTELL. PROP. L.J. 349, 350 (2002).

8. *Id.* at 350-51. For a particularly effective and comprehensive explanation of software programming technology see Vetter, *supra* note 3, at 578-82.

9. Nadan, *supra* note 7, at 350-51. "The compilation of software is an automated task that is usually carried out by standardized compiler software . . . . [C]ompilation of source code into object code is a one-way process." Strasser, *supra* note 2, ¶ 8.

10. "The different characteristics of source code and object code and their unique relationship create a powerful incentive for software producers to offer consumers only the object code of their programs and to keep the source code confidential." Strasser, *supra* note 2, ¶ 10.

11. "Most license agreements for commercial software prevent the licensee/user from having access to the source code." Dennis M. Kennedy, *A Primer on Open Source Licensing Legal Issues: Copyright, Copyleft and Copyfuture*, 20 ST. LOUIS U. PUB. L. REV. 345, 346 (2001). One reason why software developers have preferred to restrict access to source code is "[i]f software were distributed in source code form, any skilled engineer could take and reuse the innovative or labor-intensive parts of the program in that engineer's own competing program." Nadan, *supra* note 7, at 351. Nadan also discusses the fact that a software developer can seek trade secret protection for the code, but only if it remains secret. He writes, "[o]ne important way of keeping the source code secret is to distribute only object code copies of the program." *Id.* at 351-52.

12. "As a philosophical matter, computer programs were not meant to be 'owned' by a single source; as a practical matter, programs so produced and owned were likely to be functionally inferior; as a cultural matter, communication and collaboration among technologists ought not to be circumscribed." Michael J. Madison, *Reconstructing the Software License*, 35 LOY. U. CHI. L.J. 275, 284 (2003).

13. One author explains:

> For someone who simply wants to run the program, the source code is unnecessary. However, for someone who wants to do anything else, the source code is generally required. Engineers often compare having the source code to having the ability to open the hood of a car, see how the engine works, and work on it.

Stephen M. McJohn, *The Paradoxes of Free Software*, 9 GEO. MASON L. REV. 25, 27-28 (2000).

respect to software. Most software users don't care at all about what the source code looks like, they merely want to run the program. However, much like the do-it-yourself mechanics, there is a substantial (and vocal) contingent of programmers—professionals and hobbyists—that want to peek under the hood and get their hands greasy with the source code. Accordingly, open source software is developed by programmers who share their source code with each other and distribute their software with the source code freely available.

One author explains that "[a] fundamental tenet of Open Source software is that the licensee/user must get both the access to source code and, more important, the right to make changes to the source code to correct defects and bugs, customize programs or add features as the licensee/user deems appropriate."[14] Access to the source code and the ability to make changes and improvements are essential aspects of the open source approach and are what distinguish open source software from proprietary software.

III. SOFTWARE AS PROPERTY

The United States Constitution grants Congress the authority "[t]o promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries."[15] It is this constitutional authority that is the basis for intellectual (i.e., not tangible) property protection of software via copyright and patent law. This constitutional authorization reflects a primarily utilitarian justification for intellectual property protection. We do not recognize and protect intellectual property because it is the natural right of the author/inventor; we recognize intellectual property because it serves the general welfare by stimulating progress in science and the useful arts.[16] Property rights in intellectual creations provide an economic incentive for authors, inventors, and even computer programmers to create new works and inventions. It must, however, be recognized that, at least in the United States, this incentive is not an end in itself. It is merely a means to a greater end: to provide the public with creative works and useful inventions.[17] In the context of copyright protection, Justice Stewart articulated this rationale by saying:

---

14. Kennedy, *supra* note 11, at 346.

15. U.S. CONST. art. I, § 8, cl. 8.

16. For a thoughtful article considering the normative underpinnings of copyright law, see generally Jon M. Garon, *Normative Copyright: A Conceptual Framework for Copyright Philosophy and Ethics*, 88 CORNELL L. REV. 1278 (2003).

17. The utilitarian justification for intellectual property protection subscribed to by the United States stands in contrast to a natural right rationale of the continental European countries. Garon explains:

> [T]he U.S. Copyright Act is geared towards promoting innovation and a healthy information industry by providing sufficient incentives to potential creators, while at the same time preserving a "robust" public domain.
>
> . . . .
>
> Unlike the United States, continental-European "authors' rights" are based primarily on notions of natural justice: "authors' rights are not created by law but always existed in the legal consciousness of man." In the pure *droit d'auteur* philosophy, copyright is an essentially unrestricted natural right reflecting the "sacred" bond between the author and his personal creation.

Id. at 1301 (footnote omitted).

> The limited scope of the copyright holder's statutory monopoly, like the limited copyright duration required by the Constitution, reflects a balance of competing claims upon the public interest: Creative work is to be encouraged and rewarded, but private motivation must ultimately serve the cause of promoting broad public availability of literature, music, and the other arts. The immediate effect of our copyright law is to secure a fair return for an "author's" creative labor. But the ultimate aim is, by this incentive, to stimulate artistic creativity for the general public good. "The sole interest of the United States and the primary object in conferring the monopoly," [the Supreme] Court has said, "lie in the general benefits derived by the public from the labors of authors."[18]

Software is now treated as intellectual property, and as such, the owner of that property has certain property rights that are recognized and protected by law. The limited scope of that protection, however, is a reflection of the ongoing balance between the rights of the author/inventor and the public interest.

A. Copyright Protection for Computer Software

Software has not always been protected as intellectual property. Section 102 of the Copyright Act provides that "[c]opyright protection subsists . . . in original works of authorship fixed in any tangible medium of expression."[19] Historically, copyright had been used to protect authors of literary, artistic, musical and dramatic works, but in the early days of computer programming, copyright did not extend to software. As technology has progressed, however, copyright protection has been extended to new and different modes of expression, including software.[20] In 1980, Congress passed the Computer Software Copyright Act, which added a definition of "computer program" to the federal copyright statute.[21] Since that time, courts have generally agreed that computer programs are considered "literary works" and are copyright protected.[22]

The more difficult issue, however, has been determining the precise scope of that protection. At the very least, copyright protects the source code as well as the object code of a computer program.[23] But, section 102(b) of the Copyright Act specifically limits the scope of protection for all works by providing that "[i]n no case does copyright protection for an original work of authorship extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of

---

18. Twentieth Century Music Corp. v. Aiken, 422 U.S. 151, 156 (1975) (footnotes and citations omitted).

19. 17 U.S.C. § 102 (2000).

20. In 1976 Congress passed amendments to the Copyright Act. "The Act's legislative history suggested that programs were copyrightable as literary works." COHEN ET AL., COPYRIGHT IN A GLOBAL INFORMATION ECONOMY 241 (2002).

21. *See* Act of Dec. 12, 1980, Pub. L. No. 96-517, ch. 38, sec. 211, § 10, 94 Stat. 3015, 3028 (1980).

22. *See, e.g.,* Computer Assocs. Int'l, Inc. v. Altai, Inc., 982 F.2d 693 (2d Cir. 1992); Whelan Assocs., Inc. v. Jaslow Dental Lab., Inc., 797 F.2d 1222 (3d Cir. 1986); Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240 (3d Cir. 1983) (holding that copyright protection extends to both source code and object code). These cases are collected and discussed in COHEN ET AL., *supra* note 20, at 243-55.

23. "[A] computer program, whether in object code or source code, is a 'literary work' and is protected from unauthorized copying, whether from its object or source code version." Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d at 1249.

the form in which it is described, explained, illustrated, or embodied in such work."[24] Historically, distinguishing between copyrightable expression and unprotected ideas, processes, or methods of operation has proven difficult. When faced with this challenge in the seminal case of *Nichols v. Universal Pictures Corp*., Judge Learned Hand declared that "[n]obody has ever been able to fix that boundary, and nobody ever can."[25] Nonetheless, Judge Hand assumed that challenge and articulated an "abstraction" analysis to be applied on a case-by-case basis.[26] Judge Hand's abstraction test has become a standard analysis for distinguishing idea from expression in computer software.[27]

In 1986, the Third Circuit Court of Appeals was presented with a copyright infringement action concerning a software program that was written to manage a dental laboratory.[28] Because the alleged infringing program was not written in the same programming language as the original program,[29] this was not a case of literal copying. Thus, the critical question was the extent to which the original program's non-literal aspects (e.g., structure, sequence, and organization) were protected by copyright.[30] Foregoing Judge Hand's abstraction analysis, the Court instead looked to another seminal case—*Baker v. Selden*[31]—to determine the scope of copyright protection for the computer program.[32] The Court reasoned that "the line between idea and expression may be drawn with reference to the end sought to be achieved" and that "the purpose or function of a utilitarian work would be the work's idea, and everything that is not necessary to that purpose or function would be part of the expression of the idea."[33] The Court moved on to narrowly define the purpose of this particular computer program as the "efficient organization of a dental laboratory,"[34] leaving plenty of room for copyright protection of non-literal aspects of the program, such as

---

24. 17 U.S.C. § 102(b) (2000).

25. 45 F.2d 119, 121 (2d Cir. 1930).

26. Judge Hand explained that:

> Upon any work, and especially upon a play, a great number of patterns of increasing generality will fit equally well, as more and more of the incident is left out. The last may perhaps be no more than the most general statement of what the play is about, and at times might consist only of its title; but there is a point in this series of abstractions where they are no longer protected, since otherwise the playwright could prevent the use of his "ideas," to which, apart from their expression, his property is never extended.

Id.

27. *See* Bryan Seigworth, Note and Comment, *Injuring Competition and Impeding the Progress of Science: Why* Bowers v. Baystate Technologies *Was Wrongly Decided,* 23 J.L. & COM. 205, 209 (2004). *See also, e.g.,* Computer Assocs. Int'l, Inc. v. Altai, Inc. 982 F.2d 693, 706 (2d Cir. 1992) ("[T]he theoretic framework for analyzing substantial similarity expounded by Learned Hand in the *Nichols* case is helpful in the present context.").

28. *See* Whelan Assocs. v. Jaslow Dental Lab., Inc., 797 F.2d 1222 (3d Cir. 1986).

29. *Id.* at 1226.

30. "The question therefore arises whether mere similarity in the overall structure of programs can be the basis for a copyright infringement, or, put differently, whether a program's copyright protection covers the structure of the program or only the program's literal elements, *i.e.*, its source and object codes." *Id.* at 1233-34.

31. 101 U.S. 99 (1879).

32. Whelan Assocs. v. Jaslow Dental Lab., Inc., 797 F.2d 1222, 1235 (3d Cir. 1986).

33. *Id*. at 1236 (emphasis omitted).

34. *Id*. at 1240.

structure, sequence, and organization. The broad protection offered by the Third Circuit's analysis has been both applauded and criticized.[35]

Six years later, in *Computer Associates, International v. Altai, Inc.*, the Second Circuit Court of Appeals rejected the *Whelan* analysis,[36] and applied a modified abstraction test.[37] This modified test involved (1) analyzing the program from various levels of abstractions, (2) filtering out all unprotected elements (such as elements dictated by efficiency, interoperability or other external limitations), and (3) comparing the remaining parts of the original program with the allegedly infringing program for substantial similarity.[38] This analysis provides a decidedly more narrow scope of copyright protection for a program's non-literal aspects.[39]

Debating the merits of the differing approaches used by the Second and Third Circuits is beyond the purview of this paper. Nonetheless, the scope of copyright protection for computer software is a major concern for both proprietary and open source software developers. It dictates what aspects of a program are off limits as copyrightable expression, and what aspects are fair game for a future developer to copy. To some extent, the analysis of the scope of copyright protection should be informed by balancing the author's creative rights with the public interest in access. A narrower scope of protection, represented by the abstraction, filtration, comparison analysis, provides greater public access, whereas a broader scope of protection inhibits public access. For utilitarian works, such as computer software, there may be a more limited need for broad copyright protection as an incentive provider. By their very nature, utilitarian works have their own built-in incentives.

Copyright protection secures to the author of a software program a bundle of exclusive rights, including the right to make copies of the program, the right to prepare derivative works, and the right to distribute literal copies or derivative works by sale, lease or other modes of transfer.[40] These rights, however, are subject to some very

---

35. *See, e.g.,* Arthur R. Miller, *Copyright Protection for Computer Programs, Databases, and Computer-Generated Works: Is Anything New Since CONTU?*, 106 HARV. L. REV. 978, 996-98 (supporting the Third Circuit's analysis); Strasser, *supra* note 2, ¶¶ 28-29 (criticizing the Third Circuit for offering overly broad copyright protection).

36. "We think that Whelan's approach to separating idea from expression in computer programs relies too heavily on metaphysical distinctions and does not place enough emphasis on practical considerations." Computer Assocs. Int'l, Inc. v. Altai, Inc. 982 F.2d 693, 706 (2d Cir. 1992).

37. "While the abstractions test was originally applied in relation to literary works such as novels and plays, it is adaptable to computer programs." *Id.* at 706-07.

38. *Id.* at 706. Judge Walker explains the court's abstraction, filtration, comparison approach:

> In ascertaining substantial similarity under this approach, a court would first break down the allegedly infringed program into its constituent structural parts. Then, by examining each of these parts for such things as incorporated ideas, expression that is necessarily incidental to those ideas, and elements that are taken from the public domain, a court would then be able to sift out all non-protectable material. Left with a kernel, or possible kernels, of creative expression after following this process of elimination, the court's last step would be to compare this material with the structure of an allegedly infringing program. The result of this comparison will determine whether the protectable elements of the programs at issue are substantially similar so as to warrant a finding of infringement.

Id.

39. It should be noted that the Tenth Circuit Court of Appeals has followed the lead of the Second Circuit. *See* Autoskill, Inc., v. Nat'l Educ. Support Sys., Inc., 994 F.2d 1476 (10th Cir. 1993).

40. *See* 17 U.S.C. § 106 (2000).

specific statutory limitations.[41] These limitations represent a codification of the attempted balance between protection of the author's rights and the rights of the general public. They include the right to 'fair use' of a copyrighted work.[42] Section 107 of the Copyright Act provides that "the fair use of a copyrighted work, including such use by reproduction in copies or phonorecords or by any other means specified by that section, for purposes such as criticism, comment, news reporting, teaching (including multiple copies for classroom use), scholarship, or research, is not an infringement of copyright."[43] Congressional codification of the fair use doctrine signifies the belief that, in certain circumstances, society's interest in free and unrestricted access to copyrighted works outweighs the author's exclusive rights to control that work.[44]

In the context of computer software, at least some courts have held that decompilation of object code may be a 'fair use.'[45] Decompilation, as the term itself suggests, is the reversal of the process of compilation; that is, turning object code back into source code.[46] Also known as reverse engineering, this process is often critical for

---

> Subject to sections 107 through 121, the owner of copyright under this title has the exclusive rights to do and to authorize any of the following:
>
> (1) to reproduce the copyrighted work in copies or phonorecords;
> (2) to prepare derivative works based upon the copyrighted work;
> (3) to distribute copies or phonorecords of the copyrighted work to the public by sale or other transfer of ownership, or by rental, lease, or lending;
> (4) in the case of literary, musical, dramatic, and choreographic works, pantomimes, and motion pictures and other audiovisual works, to perform the copyrighted work publicly;
> (5) in the case of literary, musical, dramatic, and choreographic works, pantomimes, and pictorial, graphic, or sculptural works, including the individual images of a motion picture or other audiovisual work, to display the copyrighted work publicly; and
> (6) in the case of sound recordings, to perform the copyrighted work publicly by means of a digital audio transmission.

Id.

41. Note that section 106 of the Copyright Act begins with the phrase "[s]ubject to sections 107 through 121 . . . ." *Id.*

42. *See* 17 U.S.C. § 107 (2000).

43. *Id.*

44. Pursuant to section 107, courts are advised to consider four factors in determining whether a particular use is a fair use: "the purpose and character of the use, . . . the nature of the copyrighted work . . . the amount and substantiality of the portion used in relation to the copyrighted work as a whole, [and] the effect of the use upon the potential market for or value of the copyrighted work." *Id.* These are not the sole factors that may be considered by courts in making a fair use determination, but they are the only factors listed in section 107. Sega Enters. Ltd. v. Accolade, Inc., 977 F.2d 1510, 1522 (9th Cir. 1993) ("The statutory factors are not exclusive.").

45. *See, e.g.*, *id.* at 1527-28. ("We conclude that where [decompilation] is the only way to gain access to the ideas and functional elements embodied in a copyrighted computer program and where there is a legitimate reason for seeking such access, [decompilation] is a fair use of the copyrighted work, as a matter of law.").

46. Strasser advises that decompilation is more complicated than one might suspect. He contends that "once a piece of source code has been translated into object code, it is virtually impossible to translate it back into the original source code." Strasser, *supra* note 2, at 8. He explains:

> Software firms do offer decompilers, which attempt to reverse engineer object code, but while decompilers frequently generate source code which, when recompiled, leads to object

software programmers writing code that will interoperate with other programs.[47] By reverse engineering the protected code of a software program, the programmer reveals its technical requirements and can create a program that will meet and work with these technical requirements.[48]

Another limitation of the author's exclusive rights is the "first sale" doctrine, codified in section 109 of the Copyright Act.[49] The first sale doctrine is a specific limitation on the copyright holder's distribution rights and is rooted in a distinction between ownership of the copyright and ownership of the tangible object in which the copyright is embodied.[50] Section 109 permits owners of copyrighted works to resell or otherwise transfer their tangible copies without the permission of the copyright owner.[51] Without the first sale doctrine, such a transfer would likely be a violation of the copyright owner's exclusive distribution right.

One limitation of the author's exclusive rights is specifically applicable to computer software. Section 117 of the Copyright Act permits a software owner to make backup copies of the software by providing that "it is not an infringement for the owner of a copy of a computer program to make or authorize the making of another copy or adaptation of that computer program."[52]

Despite its limitations, at least one author suggests that copyright "is probably the most important prong of intellectual property law as far as the protection of software is concerned."[53] Mathias Strasser points out that copyright offers "numerous benefits and virtually no drawbacks" to software developers because it protects their primary interest in prohibiting "verbatim copying" and, unlike patent law, does not require them to reveal their source code.[54] The ability to withhold source code also allows software developers to use trade secret law to protect proprietary source code. Indeed, "[t]rade secret law protects any information that is secret, has economic value and is reasonably shielded from public access."[55] As long as these requirements are satisfied, trade secret law can be used to protect source code, object code, and the algorithms upon which the code is based. The combination of copyright and trade secret protection creates a relatively strong intellectual property protection regime for software developers. Software developers have added to this baseline of protection by

---

> code that is functionally equivalent with and may even look similar to the original object code, the decompiled source code invariably looks different from the original source code.

Id.

47. *See, e.g.,* Sega Enters. Ltd. v. Accolade, Inc., 977 F.2d at 1526 (noting that reverse engineering "of the object code in Sega's video game cartridges was necessary in order to understand the functional requirements for Genesis compatibility").

48. *See id.* at 1514-15.

49. 17 U.S.C. § 109 (2000).

50. *See* Madison, *supra* note 12, at 303.

51. "[T]he owner of a particular copy or phonorecord lawfully made under this title, or any person authorized by such owner, is entitled, without the authority of the copyright owner, to sell or otherwise dispose of the possession of that copy . . . ." 17 U.S.C. § 109 (2000).

52. 17 U.S.C. § 117 (2000).

53. Strasser, *supra* note 2, ¶ 24.

54. *Id.* ¶ 35.

55. *Id.* ¶ 39.

distributing software under restrictive licensing agreements, rather than outright sale or lease.[56] This strategy is discussed below.[57]

B. Patent Protection for Computer Software

Like copyright protection, patent law did not protect software until relatively recently.[58] While copyright law protects original expression, patent law provides protection for new and useful inventions.[59] Section 101 of the Patent Act provides that "[w]hoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent . . . ."[60] In the context of software programs, the crucial issue is whether the mathematical algorithms underlying the source code can be considered a "process" or whether they are merely non-patentable mathematical formulae.[61] Despite the U.S. Supreme Court's broad interpretation of this statutory language as protecting "anything under the sun that is made by man,"[62] the Court twice held that these algorithms were not patentable; first in 1972 and then again in 1978.[63]

However, in 1981 the Court reversed direction and allowed a patent for a software application that facilitated a rubber manufacturing process. On facts similar to the cases in the 1970s, the Court held in *Diamond v. Diehr*[64] that a manufacturing process

---

56. Josh Lerner & Jean Tirole, *The Scope of Open Source Licensing*, 21 J.L. ECON. & ORG. 20, 22 (2005). Lerner and Tirole explain:

> When for-profit companies manufacture proprietary software products, these copyrighted works are typically licensed rather than sold. By licensing the software, software manufacturers can limit their liability if the product does not work effectively and restrict the rights that the users would normally have (e.g., the ability to simultaneously run the software on several computers).

Id.

57. *See infra* Part IV.

58. "Through the 1970s, software was considered equivalent to mathematical algorithms or laws of nature, and thus was not patentable." David S. Evans & Anne Layne-Farrar, *Software Patents and Open Source: The Battle Over Intellectual Property Rights*, 9 VA. J.L. & TECH. 10, ¶ 8 (2004).

59. 35 U.S.C. § 101 (2000).

60. *Id*.

61. Strasser, *supra* note 2, ¶ 16. *See also* Pamela Samuelson, Benson *Revisited: The Case Against Patent Protection for Computer Programs and Other Computer-Related Inventions*, 39 EMORY L.J. 1025, 1028 n.4 (1990) ("The legal issue before the Court in Benson, and in most of the subsequent cases on the patentability of computer program-related inventions, was whether the claimed invention [an algorithm] was a 'process' that was patentable under the patent statute.").

62. Diamond v. Chakrabarty, 447 U.S. 303, 309 (1980) (quoting S. Rep. No. 1979, at 5 (1952); H.R. Rep. No. 1923, at 6 (1952)).

63. *See* Gottschalk v. Benson, 409 U.S. 63 (1972) (holding that an algorithm that converted binary-coded decimal numbers into pure binary numbers was not a patentable process within the meaning of section 101 of the Patent Act); Parker v. Flook, 437 U.S. 584 (1978) (holding that an algorithm used to update alarm limits during catalytic conversion processes was not a patentable process within the meaning of section 101 of the Patent Act). These cases are collected and discussed in, *inter alia*, Strasser, *supra* note 2, ¶¶ 16-17.

64. 450 U.S. 175 (1981). The Court noted:

> [W]hen a claim containing a mathematical formula implements or applies that formula in a structure or process which, when considered as a whole, is performing a function which the patent laws were designed to protect . . . then the claim satisfies the requirements of § 101. . . . [W]e do not view respondents' claims as an attempt to patent a mathematical

did not forfeit patent protection merely because it utilized software and mathematical algorithms. In the 1990s the Court of Appeals for the Federal Circuit (CAFC) gradually—and controversially—extended the holding of *Diamond* and the scope of patent protection for computer software.[65] In *State St. Bank v. Signature Financial Group*, the CAFC declared that an algorithm was indeed patentable subject matter because it produced a "useful, concrete, and tangible result."[66] The application of patent law to computer software—and the *State Street Bank* holding in particular—has proven to be a highly contentious topic. Patent is generally considered the most powerful intellectual property regime available.[67] A patent owner receives the exclusive rights to make, use, and sell the patented product or process for a term of twenty years.[68] This is a comprehensive grant of monopoly authority and, unlike copyright law, there is no exception for independent creators of the same or similar inventions. Yet, some commentators argue that software patents do not present the kind of threat that many critics have suggested. One author points out that "the only aspects of computer programs that may be eligible for patent protection are the algorithms on which they are based . . . . Source code and object code, by contrast, merely implement these processes and are therefore not independently patentable."[69] Although patent protection is generally considered the most powerful of the intellectual property regimes, patents do present software developers with a number of dilemmas. First, the quid pro quo of patent protection is a strict disclosure requirement that "run[s] counter to the interests of patentees generally and, in particular, to those of software developers, as they require revelation of any trade secrets that their code may contain."[70] Recall that trade secret protection is often used in conjunction with copyright law and licensing agreements to protect software developers' interests. Those that favor software patents also suggest that the limited time period of patent protection —twenty years from the date of filing the patent application—may present problems for software developers and that the costs of obtaining and enforcing patents can be prohibitively expensive.[71]

> formula, but rather to be drawn to an industrial process for the molding of rubber products . . . .

Id. at 192-93.

65. *See, e.g.,* State St. Bank & Trust, Co. v. Signature Fin. Group, Inc., 149 F.3d 1368 (Fed. Cir. 1998) (holding that a software program used in data processing was patentable); *In re* Beauregard, 53 F.3d 1583, 1584 (Fed. Cir. 1995) (vacating the Board of Patent Appeals and Interferences rejection of a patent for a computer program after the Commissioner of Patents and Trademarks agrees that "computer programs embodied in a tangible medium, such as floppy diskettes, are patentable subject matter under 35 U.S.C. § 101 . . . ." (internal quotations omitted)).

66. State St. Bank & Trust, Co. v. Signature Fin. Group, Inc., 149 F.3d at 1373 (quoting *In re* Alappat, 33 F.3d 1526, 1544 (Fed. Cir. 1994)).

67. "A valid patent provides the strongest possible protection for computer software." Carey R. Ramos & David S. Berlin, *Three Ways to Protect Computer Software*, COMPUTER LAWYER, Jan. 1999, at 16, 21.

68. 35 U.S.C. § 154 (2000).

69. Strasser, *supra* note 2, ¶ 15.

70. *Id.* ¶ 22.

71. *Id.* It is not entirely convincing that the twenty-year limitation for patent protection is a problem for software developers. The life span for software is much shorter than twenty years. If anything, the length of patent protection should be shorter for software than it is for other patented inventions.

Nevertheless, critics of software patents argue that they actually impede the progress of software innovation, contrary to the constitutional basis for our patent system.[72] Even proponents of software patents have been willing to acknowledge that "as far as patent law is concerned, there are certain concerns that the way in which the PTO [Patent and Trademark Office] and the CAFC [Court of Appeals for the Federal Circuit] currently apply it to software may hamper, rather than promote progress and innovation in the software industry."[73] Many open source proponents have been vocal critics of the extension of patent protection to computer software.[74] These critics suggest that software patenting is inherently antithetical to the open source philosophy of free and open access to software source code because patent protection precludes others from using the source code without express permission from the patent owner.[75] Open source proponents also argue that patent protection for software is unnecessary and may even inhibit innovation in software technology.[76] Indeed, software innovation was occurring at a rapid pace long before patent protection was extended to software.[77] Nevertheless, proponents of software patents insist that the software industry has fundamentally changed and contend that patent protection is now a necessary intellectual property right to spur innovative software development.[78]

The dispute over software patent protection reflects a fundamental question embedded within the open source debate. To what extent are the incentives provided by intellectual property protection necessary to promote progress and innovation in the field of computer software development? And, does the open source movement require a recalibration of the scales in striking a balance between protecting the author/inventor and ensuring public access? Open source advocates suggest that the relative success of the open source approach reveals that exclusive intellectual property protection of software as an incentive to promote progress and innovation is unnecessary.[79] They argue that under an open source approach, the benefits of collaboration more than make up for any lost economic incentives.[80] Indeed, this faith in the open source approach is pervasive in the academic literature.

Nevertheless, at least one author is not so enamored with the open source approach. Mathias Strasser argues that "[t]he fundamental problem associated with all of the various policy arguments [supporting the open source approach] is that the

---

72. *See* Evans & Layne-Farrar, *supra* note 58, ¶ 54 (discussing and responding to the argument that patent thickets will discourage innovation).

73. Strasser, *supra* note 2, ¶ 45.

74. *See* Evans & Layne-Farrar, *supra* note 58, ¶ 1.

75. Critics of copyright protection for software made these same claims, *id.*, but copyright law is now an integral part of the open source approach. One wonders whether the open source movement can use patent law in ways that promote open source development, similar to the use of copyright law. Could open source developers build an arsenal of patents and make them freely available? Probably not. Unlike copyright, patent does not vest in the inventor as soon as the invention is complete. There is a lengthy and expensive patent prosecution process that is required before a patent is given. The costs and the lengthy process serve as a barrier to open source participation.

76. *Id.* ¶¶ 44, 54.

77. *Id.* ¶ 45.

78. *See id.* ¶¶ 47-51.

79. *See, e.g., infra* note 176.

80. *See, e.g.*, *infra* note 176.

cooperative spirit fueling the development of the open source software is not a viable substitute for the market."[81] He suggests that a purely open source software development model would fail to provide the kinds of economic incentives necessary to promote sustainable progress and innovation of software technology.[82] Strasser does recognize that the open source development model offers some benefits over the proprietary model, but believes that absent the economic incentives of a proprietary system, progress would be inhibited. He insists that "[s]ince open source software does not seem to provide developers with sufficiently powerful incentives to write enough code of the type that consumers want, it would be undesirable to mandate that all software be legally open."[83]

Strasser's observations are consistent with the economic incentive based theoretical underpinnings of our intellectual property regime. These assumptions, however, are challenged by open source proponents such as Eben Moglen, Professor of Law at Columbia University and general counsel for the Free Software Foundation (FSF).[84] "Moglen objects to the conventional economic premise that people are motivated only by incentives without which they will not engage in productive activity like writing software code."[85] Rather, he believes that software innovation is an emergent property of connected human beings.[86] Although Moglen is probably correct to some extent that there are programmers who have sufficient incentive to write code without economic reward, it is unclear how generalizable and sustainable this is. Economic incentive has always been and will continue to be an important tool to foster and reward desirable activities.

IV. SOFTWARE LICENSING

Although software is protected by copyright, trade secret, and patent law, software developers are justifiably skeptical of the intellectual property regime's ability to adequately protect their interests. Given the digital nature of software, it is particularly vulnerable to widespread copying. Software can be copied many times over without any functional degradation of the program. Additionally, interconnectivity of software users through the Internet facilitates rampant copying of software programs across space and time. Historically, software distributors have relied on contract law—in the form of licensing agreements—to extend the protection offered by intellectual property.[87]

---

81. Strasser, *supra* note 2, ¶ 78.

82. *Id.* ("The prospect of not being able to make money off of software would undermine the monetary incentives which, in a market economy, serve the important function of inducing software producers to manufacture software in the first place.").

83. *Id.* ¶ 86. With regard to this last assertion, I would tend to agree with Strasser that mandating all software be open source is not a good idea. No doubt, many within the open source community would support this approach, but such an all-or-nothing view strikes me as overly principled and not very pragmatic. It is possible to promote and foster the open source movement without mandating such a paradigm shift. For a discussion of several ways in which governments can support open source, see generally Potter, *supra* note 6.

84. *See, e.g.*, *infra* note 176.

85. David McGowan, *Legal Implications of Open-Source Software*, 2001 U. ILL. L. REV. 241, 266.

86. *See infra* note 176.

87. *See* Ravicher, *supra* note 4, ¶ 34.

The standard model for software development and distribution has evolved over the brief history of the enterprise. In the early days, software programmers were not writing code for mass-market software programs. There was no mass market for software during the 1970s because personal computing had not yet emerged.[88] Software programmers usually contracted with specific clients to write code for a particular purpose.[89] At that time it was still unclear whether software programmers could utilize copyright or patent protection for their work; programmers, instead, relied on trade secret and contract law.[90] Even after it became clear that software was protected by copyright, software developers retained the licensing approach. In *Softman Products Co. v. Adobe Systems, Inc.*, Judge Pregerson explained the evolution of the licensing approach this way:

> Historically, the purpose of "licensing" computer program copy use was to employ contract terms to augment trade secret protection in order to protect against unauthorized copying at a time when, first, the existence of a copyright in computer programs was doubtful, and, later, when the extent to which copyright provided protection was uncertain. [91]

As personal computing emerged, the growing demand for mass-market software applications required a new licensing model.[92] No longer could software programmers rely on negotiating specific license terms with individual clients. Thus, the mass-market software license was born.

As mentioned above, software licenses help developers and distributors bolster the baseline protection afforded by intellectual property law. From the perspective of the software licensor, licensing transactions are preferable to sales, leases or other modes of transfer for a number of reasons. First, a license allows the distributor to maintain his or her intellectual property rights in the program and grant only those rights specified in the license.[93] For example, a licensor could permit a licensee to make copies of a program, but restrict any rights to distribute those copies. Thus, a license allows the licensor to impose restrictions and collect royalties based on usage.[94] Second, licenses provide software developers with the flexibility of granting an exclusive license to a single licensor or nonexclusive licenses to several or many licensors.[95] Licensing also provides a potential end-run around statutory limitations to the copyright holder's exclusive rights discussed above.[96]

---

88. Most of the software written at this time was for large mainframe computers. *See* Robert W. Gomulkiewicz & Mary L. Williamson, *A Brief Defense of Mass Market Software License Agreements*, 22 RUTGERS COMPUTER & TECH. L.J. 335, 338-39 (1996).

89. Ira V. Heffan, Note, *Copyleft: Licensing Collaborative Works in the Digital Age*, 49 STAN. L. REV. 1487, 1494 (1997).

90. *Id.*

91. 171 F. Supp. 2d 1075, 1083 (C.D. Cal. 2001).

92. Gomulkiewicz & Williamson, *supra* note 88, at 339.

93. Peter Quittmeyer, *Software Licensing*, 2003 PRACTISING L. INST. 903, 909, *available at* WL 763 PLI/PAT 903 ("In general, a license does not grant or create a property interest at all, but merely a permission to act or, conversely, a covenant not to sue for such action.").

94. *Id*. Presumably, the more rights that are transferred, the higher the royalty rate.

95. Heffan, *supra* note 89, at 1496.

96. *See* William H. Neukom & Robert W. Gomulkiewicz, *Licensing Rights to Computer Software*, 1993 PRACTISING L. INST. 775, 777-78, *available at* WL 354 PLI/PAT 775 (suggesting that software

Of particular concern to software distributors is the 'first sale' doctrine.[97] Under this doctrine—codified as section 109(a) of the Copyright Act—once a copy of a copyrighted work has been sold, the copyright owner loses authority to control the subsequent transfer of that particular tangible copy.[98] Because software can be easily copied with no degradation to the program, the 'first sale' doctrine could potentially justify and facilitate widespread infringement. Initially, courts were reluctant to interpret license agreements as anything other than a sale.[99] More recently, however, some courts have held that a software license is not a sale and therefore the first sale doctrine is inapposite.[100]

Software licensors have also maintained that the licensing of software is not a 'sale of goods' and therefore should not be governed by Article 2 of the Uniform Commercial Code (UCC).[101] Beyond concerns about the first sale doctrine, software developers are keenly interested in avoiding any implied warranties and liabilities provided by the UCC.[102] Contrary to the desires of software developers and distributors, however, courts have generally held that software license transactions are covered by Article 2—either directly or by analogy—and thus are subject to implied warranties and liabilities.[103] Nevertheless, courts have not offered entirely consistent guidance in this area.[104]

A. Shrinkwrap and Clickwrap Licenses

The legitimacy of the mass-market software licensing approach, in general, and the validity of some license terms, specifically, has not gone unchallenged. Software developers utilize standard form mass-market license agreements to give them the same kind of protection and control that contract law affords them. But a license agreement

---

publishers license software to, *inter alia*, negate the first sale doctrine under the Copyright Act).

97. *Id.*

98. "[T]he owner of a particular copy . . . is entitled, without the authority of the copyright owner, to sell or otherwise dispose of the possession of that copy. . . ." 17 U.S.C. § 109(a) (2000).

99. *See, e.g.*, Step-Saver Data Sys., Inc. v. Wyse Tech., 939 F.2d 91 (3d Cir. 1991).

100. *See, e.g.*, Adobe Sys. Inc. v. One Stop Micro, Inc., 84 F. Supp. 2d 1086, 1092 (N.D. Cal. 2000) (holding that the distribution agreement was a license and not a sale, therefore, the 'first sale' doctrine was inapplicable); *but see* Softman Products Co. v. Adobe Sys., Inc., 171 F. Supp. 2d 1075, 1085 (C.D. Cal. 2001) (software transaction interpreted as a sale instead of a license because "the purchaser commonly obtains a single copy of the software, with documentation, for a single price . . . [and] [t]he license runs for an indefinite period."). These cases are collected and discussed in, *inter alia*, Quittmeyer, *supra* note 93, at 911.

101. This contention is comprised of two arguments. First, a license is not a 'sale.' Second, software is not a 'good.'

102. *See* Neukom & Gomulkiewicz, *supra* note 96, at 777-78 ("Most software publishers offer a limited warranty and then disclaim other warranties in accordance with the provisions of the Uniform Commercial Code."). See UCC implied warranties at U.C.C. §§ 2-312 to 2-316 (2004).

103. "Regardless of the software's specific form or use, it seems clear that computer software, generally, is considered by the courts to be a tangible, and movable item, not merely an intangible idea or thought and therefore qualifies as a 'good' under Article 2 of the UCC." Commc'ns Groups, Inc. v. Warner Commc'ns, Inc., 527 N.Y.S.2d 341, 344 (N.Y. Civ. Ct. 1988) (citations omitted). *See also* I.Lan Sys., Inc. v. Netscout Serv. Level Corp., 183 F. Supp. 2d 328, 332 (D. Mass. 2002) ("Article 2 technically does not, and certainly will not in the future, govern software licenses, but for the time being, the Court will assume it does.").

104. *See supra* note 103.

is not necessarily a contract.[105] In the context of software licenses, unlike a traditional contract, there really is no bargained-for exchange.[106] Software distributors rely on a "notice-plus-conduct model"[107] of contract formation. The software provider essentially makes an offer on the terms stated in the license agreement (notice) and the licensee assents to those terms by using the software (conduct). Notice of the terms of the offer, prior to assent, is crucial to this theory of contract formation.[108] This notice-plus-conduct model generally takes two forms.

Software developers first relied on so-called "shrinkwrap" licenses to impose contractual obligations upon licensees.[109] A shrinkwrap license is so named because the software is packaged so that the terms of the agreement are visible through the shrinkwrap covering on the box, or the package indicates that use of the software is conditioned on terms contained inside the package.[110] A purchaser that wishes not to enter into a contract on the terms stated in the license may return the software.[111] Breaking open the shrinkwrap package and using the software or even failing to return it within a specified period indicates assent to the terms of the agreement and formation of a contract.[112] That, at least, is the hope of the software licensor.

More recently, software developers have engaged in online distribution of software via so-called "clickwrap" agreements. In a typical clickwrap transaction, the purchaser is presented with a standard form license agreement that requires the consumer to click an "I Agree" button before the software can be used or downloaded.[113] Courts have been split as to the contractual enforceability of these types of

---

105. *See* LAWRENCE ROSEN, OPEN SOURCE LICENSING: SOFTWARE FREEDOM AND INTELLECTUAL PROPERTY LAW 53 (2005) ("Open source licenses, it turns out, can be both bare licenses and contracts.").

106. *See, e.g.*, Garry L. Founds, Note, *Shrinkwrap and Clickwrap Agreements: 2B or Not 2B?*, 52 FED. COMM. L.J. 99, 100 (1999) (noting that one problem with most mass market software licenses is "the public is powerless to negotiate" the terms).

107. I borrow the phrase "notice-plus-conduct" from David McGowan. *See* McGowan, *supra* note 85, at 289. UCITA could resolve this uncertainty by declaring rules for determining when a license has the force of contract and when it does not.

108. "Simply stated, a user is not bound by a contract of which he is not made aware." Nadan, *supra* note 7, at 364.

109. *See* Heffan, *supra* note 89, at 1498 ("As the commercial software market expanded and the negotiation of individual software licenses became less practical, developers began using 'shrinkwrap' license agreements in an attempt to delineate the rights afforded their retail customers.").

110. *Id. See also*, Terry J. Ilardi, *Mass Licensing--Part 1: Shrinkwraps, Clickwraps and Browsewraps*, 2005 PRACTISING L. INST. 251, 256, available at WL 831 PLI/PAT 251 ("[M]any software vendors merely indicate on the outside of the packaging that the software is licensed according to a license contained within the package.").

111. Neukom & Gomulkiewicz, *supra* note 96, at 777.

112. Heffan, *supra* note 89, at 1498.

113. *See, e.g.*, I.Lan Systems, Inc. v. Netscout Service Level Corp. 183 F. Supp. 2d 328, 329 (D. Mass. 2002). Judge Young begins his opinion with this question:

> Has this happened to you? You plunk down a pretty penny for the latest and greatest software, speed back to your computer, tear open the box, shove the CD-ROM into the computer, click on "install" and, after scrolling past a license agreement which would take at least fifteen minutes to read, find yourself staring at the following dialog box: "I agree." Do you click on the box? You probably do not agree in your heart of hearts, but you click anyway, not about to let some pesky legalese delay the moment for which you've been waiting.

Id.

agreements, as well as the proper mode of analysis. Most courts have analyzed these agreements according to the provisions of the UCC, while a minority of courts has found that the UCC is inapplicable to these types of licensing transactions.[114]

In *ProCD, Inc. v. Zeidenberg*,[115] the Seventh Circuit held that a shrinkwrap license was indeed a contract and was therefore governed by the UCC. ProCD, the plaintiff, had compiled a database that included information from over 3000 telephone directories.[116] Zeidenberg purchased the database software, in CD-ROM form, from a retail store and made the information available on the Internet for a fee.[117] Zeidenberg's publication of the database information on the Internet was in direct violation of the license terms.[118] The box containing the CD-ROM indicated that the use of the software was restricted by the terms of the enclosed license.[119] The license terms were included on a printed manual inside the box and were displayed on the computer screen every time the software application was run.[120] Writing for the majority, Judge Easterbrook held that "[s]hrinkwrap licenses are enforceable unless their terms are objectionable on grounds applicable to contracts in general."[121]

One year later, in *Hill v. Gateway 2000, Inc*.,[122] the Seventh Circuit applied and extended the *ProCD* holding. Hill arranged for the purchase of a Gateway computer—with software pre-installed—by providing a credit card number over the telephone.[123] When the computer arrived, the box did not provide notice of additional licensing terms contained inside, unlike the package in *ProCD*.[124] Despite the lack of external notice, the court held that the Hills did have notice of the terms of the agreement and were, therefore, bound by the arbitration clause they sought to avoid.[125]

---

114. Kevin W. Grierson, Annotation, *Enforceability of "Clickwrap" or "Shrinkwrap" Agreements Common in Computer Software, Hardware, and Internet Transactions*, 106 A.L.R.5th 309 §2(a)(2003).

115. 86 F.3d 1447, 1450 (7th Cir. 1996) ("[W]e treat the licenses as ordinary contracts accompanying the sale of products, and therefore as governed by the common law of contracts and the Uniform Commercial Code."). Interestingly, the court went on to remark that "[w]hether there are legal differences between 'contracts' and 'licenses' (which may matter under the copyright doctrine of first sale) is a subject for another day." *Id*.

116. *Id*. at 1449.

117. *Id*. at 1450.

118. *See id.* Some readers might question whether Zeidenberg's unauthorized copying of the information also constitutes copyright infringement. It does not. In the seminal case of *Feist Publications, Inc. v. Rural Telephone Service Co.*, the Supreme Court held that the compilation of names and telephone numbers in a phone book is not an original work of authorship within the meaning of section 102 of the Copyright Act. 499 U.S. 340, 362 (1991). *ProCD* illustrates the added protection that a licensing agreement provides when copyright protection is insufficient to protect a particular work.

119. ProCD, Inc. v. Zeidenberg, 86 F.3d at 1450.

120. *Id*.

121. *Id*. at 1449.

122. 105 F.3d 1147 (7th Cir. 1997).

123. *Id*. at 1148.

124. *Id*. at 1150. Apparently this argument was raised for the first time at oral argument. *See id*. Judge Easterbrook explained that this distinction was merely functional and had no legal significance. *Id*.

125. *Id*. at 1151. "[T]he Hills knew before they ordered the computer that the carton would include *some* important terms, and they did not seek to discover these in advance." *Id*. at 1150. Let it not be assumed that enforcement of shrinkwrap licenses is confined to the Seventh Circuit and Judge Easterbrook. In a more recent case, the Washington Supreme Court upheld a shrinkwrap license—located on the outside of the box—and enforced the disclaimer of liability for consequential damages. *See* M.A. Mortenson Co. v.

In writing for the court, Judge Easterbrook relied at least in part on the impracticalities of requiring full disclosure of terms prior to contract formation.[126] This rationale becomes important in thinking about whether it is appropriate to extend this holding to on-line transactions where disclosure of terms prior to formation is much more practical.

Courts have generally been more amenable to enforcing clickwrap licenses than shrinkwrap licenses because affirmative assent is usually given—by clicking an "I Agree" dialogue box—after notice of the terms are displayed on the computer screen and before the software can be downloaded or used.[127] As shrinkwrap and clickwrap licensing agreements have become a customary and accepted practice in the software industry, courts have shown an increasing willingness to enforce them, provided they meet the minimal requirements necessary for contract formation.[128] Enforcement is more likely where the licensor provides prominent notice of the terms and the licensee manifests affirmative assent to those terms.[129] However, different courts have employed different modes of analysis and reached inconsistent conclusions.[130]

V. THE RISE AND FALL OF UCITA

In the early 1990s an American Bar Association (ABA) Study Committee responded to the uncertainties regarding software licensing transactions and the emerging case law by recommending the development of a uniform law to govern computer software transactions.[131] The NCCUSL and the American Law Institute (ALI) both agreed at the time that a uniform code would help bring certainty and consistency to the rules applicable to software transactions.[132] As a commercial code, UCITA is

---

Timberline Software Corp., 998 P.2d 305, 307-08 (Wash. 2000). Relying on the UCC, the court found that the parties' previous course of dealing, as well as trade usage, supported enforcement of the license and the liability disclaimer. *Id*. at 314.

126. *See* Hill v. Gateway 2000, Inc., 105 F.3d at 1149. Judge Easterbrook explained:

> Payment preceding the revelation of full terms is common for air transportation, insurance, and many other endeavors. Practical considerations support allowing vendors to enclose the full legal terms with their products. Cashiers cannot be expected to read legal documents to customers before ringing up sales. If the staff at the other end of the phone for direct-sales operations such as Gateway's had to read the four-page statement of terms before taking the buyer's credit card number, the droning voice would anesthetize rather than enlighten many potential buyers. Others would hang up in a rage over the waste of their time. And oral recitation would not avoid customers' assertions (whether true or feigned) that the clerk did not read term X to them, or that they did not remember or understand it.

Id.

127. *See, e.g.*, I.Lan Sys., Inc. v. Netscout Serv. Level Corp., 183 F. Supp. 2d 328, 338 (D. Mass. 2002) ("If *ProCD* was correct to enforce a shrinkwrap license agreement, where any assent is implicit, then it must also be correct to enforce a clickwrap license agreement, where the assent is explicit.").

128. *See* Ilardi, *supra* note 110, at 273 ("While questions remain as to the enforceability of properly drafted shrinkwrap, clickwrap or even browserwrap agreements, the majority of jurisdictions have, *particularly in the last few years*, held that most such agreements are enforceable.") (emphasis added).

129. *See id.*

130. *See* Grierson, *supra* note 114.

131. Irene Kosturakis, *Software Licensing and UCITA*, 2003 PRACTISING L. INST. 437, 444 *available at* WL 762 PLI/PAT 437.

132. *Id.* at 445.

rooted on the premise that clarity, uniformity, stability and freedom to contract are virtues in a legal system that seeks to foster commercial development.[133] After some initial disagreement about whether this new law would be incorporated into the existing Article 2 of the UCC, the NCCUSL Executive Committee concluded that a separate Article 2B was the best approach, given the inherent distinctions between the sale of goods and the licensing of information rights.[134] At the time, some believed that with respect to commercial contract law, there was sufficient overlap between goods transactions and information transactions to justify inclusion in Article 2.[135] One author suggests that the decision to promulgate a new Article 2B was not dictated so much by a recognition of profound and inherent distinctions between sales of goods and licensing of computer information, but rather was influenced by pressure from the software industry seeking fundamentally different treatment than Article 2 provided.[136]

In keeping with the requirements for all uniform codes, a draft version of Article 2B was jointly developed by the NCCUSL and the ALI. ALI support for the project soon diminished, however, and in 1999 the two bodies discontinued their joint drafting efforts.[137] A press release announced that the ALI had "significant reservations about . . . some of [Article 2B's] key substantive provisions and its overall clarity and coherence."[138] Without the approval of the ALI, the proposed code was no longer eligible for enactment as part of the UCC.[139] Nonetheless, the NCCUSL continued its efforts independently and completed the project as a separate uniform law, resulting in the Uniform Computer Information Transactions Act (UCITA).[140] UCITA was officially promulgated as a model statute for independent consideration by the states on July 29, 1999.[141] The conflict between the ALI and the NCCUSL was only a sign of things to come; UCITA has been controversial from the very beginning.

Opposition to UCITA has been significant, both in terms of the broad spectrum of interests aligned against the proposed statute, as well as the passion coming from certain constituencies. Critics of UCITA have included state officials, federal agencies, consumer advocacy groups, library organizations, legal academics, large software purchasers (particularly in the insurance industry), and even software developers. In 1999, the NCCUSL received letters signed by twenty-four state Attorneys General opposing UCITA.[142] Likewise, a lack of support from the American

---

133. *Id.* at 446.

134. *Id*. at 445. Note that Article 2A of the UCC governs the leasing of goods.

135. Amelia H. Boss, *Taking UCITA on the Road: What Lessons Have We Learned?*, 2001 PRACTISING L. INST. 121, 131-32, *available at* WL 673 PLI/PAT 121.

136. *See id*. at 132-33.

137. Daniel A. DeMarco & Christopher B. Wick, *Now UCITA, Now You Don't: A Bankruptcy Practitioner's Observations on the Proposed Uniform Computer Information Transactions Act*, AM. BANKR. INST. J., May 2004.

138. *Article 2B is Withdrawn from U.C.C. and Will Be Promulgated by NCCUSL as Separate Act*, THE ALI REPORTER, Spring 1999, *available at* http://www.ali.org.

139. *See* Boss, *supra* note 135, at 135.

140. DeMarco & Wick, *supra* note 137.

141. *Id*.

142. The letter stated:

> We believe the current draft puts forward legal rules that thwart the common sense expectations of buyers and sellers in the real world. We are concerned that the policy choices embodied in these new rules seem to almost invariably favor a relatively small

Bar Association (ABA) was suspected and revealed when the NCCUSL failed to submit a resolution to the House of Delegates of the ABA to ratify UCITA in 2000.[143] Even the Federal Trade Commission expressed concern about the lack of consumer protection in UCITA and discussed possible remedies.[144]

In general terms, UCITA was viewed by critics as unduly deferential to the interests of software developers/licensors over licensees.[145] Much like the UCC, UCITA embraces and is built on the concept of freedom of contract.[146] UCITA is really a set of default rules that will apply only if the parties do not otherwise agree to different terms. Practically speaking, this means that parties with greater bargaining leverage benefit from both the UCC and UCITA because the terms of the agreement control.[147] Critics have argued first, that the freedom to contract model as adopted in UCITA is inappropriate for software licensing transactions,[148] and second, that several of UCITA's default provisions are too favorable to software developers.

The view that statutory deference to freedom of contract leads to unfair or unjust agreements is not new. Indeed, the debate over *regulation* of business transactions versus *freedom* of contract occurred over fifty years ago during the UCC drafting and redrafting process.[149] One author recalls that "[e]arly drafts of the UCC were decidedly more regulatory and included measurably more judicial oversight of commercial activities."[150] In response to criticism from commercial and business entities, the drafters modified the UCC by embracing the freedom of contract model and adopting a decidedly more *laissez-faire* approach.[151] This strategy ultimately ensured support of the business and commercial interests that would make or break state adoption.[152] Ironically, the similar approach taken in UCITA (embracing the freedom of contract) has been one of the primary roadblocks to more widespread acceptance. The same business and commercial interests that favored contract freedom in the context of the sale of goods are not so enthusiastic in the context of software licensing. As software licensees, these entities—with the exception of the most large and powerful companies—do not have the kind of bargaining leverage that will enable them to negotiate favorable terms with software licensors. With regard to UCITA, it is essentially only the software developers that have an interest in contract freedom: the rest of the

---

> number of vendors to the detriment of millions of businesses and consumers who purchase computer software and subscribe to internet services.

Letter from W.A. Drew Edmonson et al., State Attorneys General, to Gene Lebrun, NCCUSL (July 23, 1999), *available at* http://www.jamesshuggins.com/h/tek1/ucita_ags_19990723_letter.htm. Thirteen attorneys general signed this letter and eleven signed a subsequent letter in agreement. *Id.*

143. Boss, *supra* note 135, at 134 n.15.

144. *Id*. at 135 n.17.

145. DeMarco & Wick, *supra* note 137.

146. *See* Nim Razook, *The Politics and Promise of UCITA*, 36 CREIGHTON L. REV. 643, 647 (2003).

147. *Id.* at 647-48 ("If the law supports standard contracts based loosely on a freedom of contract principle, then the terms of that contract will tend to favor the drafter over the other side . . . .").

148. *Id*. at 651.

149. *Id*.

150. *Id*. at 657.

151. *Id*.

152. For a comprehensive and thoughtful discussion of the history of the drafting, redrafting, and adoption of the UCC, see Allen R. Kamp, *Downtown Code: A History of the Uniform Commercial Code 1949-1954*, 49 BUFF. L. REV. 359 (2001).

world—including consumers, commercial and business entities, and others—would prefer more of a regulatory approach. Commercial and business entities are the key players here. Without their support, adoption of UCITA will remain an uphill struggle.[153]

Some opponents have also offered the view that, at present, commercial practices and customs in the context of software licensing are not sufficiently well-developed or widely-accepted to justify codification into uniform law.[154] The software licensing regime is, indeed, a relatively nascent one. Perhaps it is too early to encode a uniform set of rules on this emerging commercial practice. Or perhaps, a state-by-state approach will offer valuable experimentation in this area of the law until a more robust set of customs and rules are established.[155] One might even argue that the open source movement represents compelling evidence of the shifting sands of commercial software development and distribution practices; evidence that could be used to support a wait-and-see approach. This view, however, fails to consider that the open source movement actually embraces—perhaps grudgingly—the copyright-then-license regime used by the commercial proprietary software developers. This concession by open source developers could be seen as a final affirmation of the copyright-then-license paradigm. Moreover, there is little value in experimenting with an ad-hoc state-by-state approach in this area given the ubiquity of software licensing in commercial transactions.

In addition to the general view that licensing of software information has not yet fully developed consistent, broadly accepted commercial practices, opponents of UCITA have challenged several specific provisions of the code as overly protective of the interests of major software developers. UCITA's validation of shrinkwrap and clickwrap license agreements was particularly troublesome to many critics. One author recalls that "the continuing refrain heard from opponents is that UCITA validates fictional assent and thus there is no contract in a classical sense."[156] This fictional assent usually comes in the form of a click on an "I agree" dialogue box. It is questionable whether most computer users read the terms of the licensing agreement or even realize that they are entering a binding agreement at all. Additionally, UCITA's validation of "pay now, terms later" licensing agreements does not seem consistent with the realities of online transactions. In the early cases involving over-the-phone software purchases and shrinkwrap licenses,[157] the courts were willing to stretch traditional contract theory to enforce terms of an agreement that were not disclosed until after payment occurred. This was justified in part on the realities of over-the-phone transactions where disclosure of all the terms is not practical.[158] But online transactions are quite different in this respect. By its very nature, the Internet makes

---

153. Professor Razook points out that "the U.C.C., as proposed to the states, enjoyed the support of one key constituency already alienated by the [NCCUSL's] proposed UCITA: business and commercial interests." Razook, *supra* note 146, at 656.

154. *See infra* note 250 and accompanying text.

155. Razook, *supra* note 146, at 659 ("The appeal of a state-by-state evolution of rules for software transactions is based on Justice Brandeis' often-used description of states as laboratories for the creation, testing, and acceptance or rejection of applicable laws.").

156. Boss, *supra* note 135, at 147-48.

157. *See supra* notes 88-101 and accompanying text.

158. *See supra* note 101 and accompanying text.

it possible to disclose all terms prior to formation of the contract. This ease of information exchange undermines the rolling contract rationale used by some courts to justify and enforce shrinkwrap license agreements. Although a *new contract* model might become necessary in the digital/internet age, the transition is disconcerting to many.

UCITA's treatment of warranties and liabilities, licensor self-help, and licensee reverse engineering has also been particularly controversial.[159] As drafted, UCITA permits software licensors to disclaim any implied warranties of merchantability or fitness and shift the risk of liability for damages to the licensee.[160] Although this is consistent with the UCC approach in the sale of goods context, some have criticized this model as overly "goodcentric."[161] With respect to a sale of goods, the purchaser conceivably has an opportunity to inspect the product before accepting the deal, whereas licensees of digital information do not have this opportunity.[162] Inspection of the product takes place only after the license agreement is binding.[163] This distinction makes warranty and liability disclaimers particularly risky for the licensee. At least one author has suggested an alternative warranty disclaimer regime where the licensor and the licensee share this risk.[164]

Probably the most troubling aspect of UCITA, however, was its treatment of licensor self-help and licensee reverse engineering. As initially drafted, UCITA permitted a software licensor to electronically disable a licensee's ability to use the software in the case of a breach of the license agreement, provided certain notice requirements were satisfied.[165] In essence, this provision gave the software licensor the authority to impose his or her own preliminary injunction against a licensee without any judicial process. The burden would then shift to licensees to show that they had not in fact breached their license. This provision is a primary example of why many UCITA opponents viewed the proposed statute as a coup for software developers. Likewise, UCITA's proscription against reverse engineering of software source code by licensees encountered similar resistance. UCITA permitted this restriction because of its deference to, and enforcement of, the terms of the licensing agreement. UCITA would have validated a provision in a licensing agreement that prohibited reverse engineering. As explained above, reverse engineering is a valuable and widely-used process in the software development industry and some courts have held that it can be considered a fair use under the Copyright Act.[166] Vehement opposition to these two

---

159. *See infra* text accompanying notes 280-81 (Richard Stallman expressing concerns with liability limitations and reverse engineering).

160. *See* § 406 of UCITA, *available at* http://www.law.upenn.edu/bll/ulc/ucita/2002final.htm.

161. Razook, *supra* note 146, at 651.

162. *Id*.

163. *Id*.

164. *See generally* Leo L. Clarke, *Performance Risk, Form Contracts and UCITA*, 7 MICH. TELECOMM. & TECH. L. REV. 1 (2001).

165. Razook, *supra* note 146, at 648-49.

166. *See, e.g.*, Sega Enters. Ltd. v. Accolade, Inc., 977 F.2d 1510, 1527-28 (9th Cir. 1993) ("We conclude that where [reverse engineering] is the only way to gain access to the ideas and functional elements embodied in a copyrighted computer program and where there is a legitimate reason for seeking such access, [reverse engineering] is a fair use of the copyrighted work . . . .").

issues, in particular, has ultimately forced the NCCUSL to amend UCITA to eliminate the self-help provision and the outright prohibition of reverse engineering.[167]

Thus far, the broad opposition and intensive lobbying efforts have limited adoption of UCITA to only two states (Virginia and Maryland).[168] In fact, twice as many states (Vermont, Iowa, North Carolina, and West Virginia) have passed anti-UCITA "bomb shelter" legislation that would preclude application of UCITA by choice-of-law licensing terms.[169] In Maine, UCITA was introduced for consideration as LD 1324 in 2001, but never escaped the House floor.[170] Despite the recent amendments adopted by the NCCUSL, Maine has not seriously reconsidered adopting UCITA. At present, it appears that UCITA has completely lost any traction and some have speculated about UCITA's demise.[171] However, such speculation may be exaggerated.[172] The original motivation for developing a uniform code to cover software transactions was a growing recognition that the sale of goods provisions of Article 2 were not well suited to software licensing and that courts were not entirely consistent in dealing with these new transactions. In this respect, nothing has changed since the ALI and the NCCUSL first conceived of a new uniform code in the early 1990s. On the other hand, the landscape of commercial software development and distribution has been altered significantly since then by the emerging commercial presence of open source software.

As states, and maybe even the ALI, consider whether or not to give UCITA another look, it might be important to think about how it would affect the open source movement. In rethinking UCITA, there would be little need to consider its implication with regard to open source software if the open source movement is merely a passing fancy. If, however, the open source approach is here to stay, it becomes an important consideration when thinking about rules that govern software licensing transactions.

---

167. As amended, section 118 of UCITA now reads:

> Notwithstanding the terms of a contract subject to this [Act], a licensee that lawfully obtained the right to use a copy of a computer program may identify, analyze, and use those elements of the program necessary to achieve interoperability of an independently created computer program with other programs, including adapting or modifying the licensee's computer program, if:
>
> (1) the elements have not previously been readily available to the licensee;
>
> (2) the identification, analysis, or use is performed solely for the purpose of enabling such interoperability; and
>
> (3) the identification, analysis, or use is not prohibited by law other than this [Act].

UCITA § 118, *available at* http://www.law.upenn.edu/bll/ulc/ucita/2002final.htm. Adoption of these amendments in 2002 has not yet stimulated any renewed interest in the statute.

168. Deanna Allen & Jon Grossman, *Is Death Knell for UCITA the Last Word*, 229 THE LEGAL INTELLIGENCIER 78 (2003).

169. Carol Ebbinghouse, *UCITA Stopped, but Librarians and Consumers Remain Vigilant*, Aug. 11, 2003, http://www.infotoday.com/newsbreaks/nb030811-1.shtml. A software licensor that wants the protection of UCITA might include in their license a provision that states that the agreement is covered by Maryland or Virginia law. This type of choice-of-law provision is not uncommon. However, in those states that have passed anti-UCITA bomb shelter legislation, this kind of choice-of-law provision would be unenforceable. *See* Boss, *supra* note 135, at 133.

170. L.D. 1324 (120th Leg. 2001).

171. *See*, *e.g.*, Allen & Grossman, *supra* note 168; Ilardi, *supra* note 110, at 255 (noting that "for all intents and purposes UCITA has failed").

172. Razook, *supra* note 146, at 667.

The next section examines the development of the open source movement, from a limited tradition within academic circles to a mainstream commercial success story. This section is included to provide some background, and more importantly, to offer a foundation for speculating about where the open source movement is going.

VI. A HISTORY OF THE DEVELOPMENT OF THE OPEN SOURCE APPROACH

A. Academic Roots

The history of the open source approach is contemporaneous with the history of software programming itself. That is to say that the idea of sharing source code is not a new one. Most accounts of the history of the open source approach begin with an MIT computer programmer named Richard Stallman.[173] According to the founding myth of the open source movement, Stallman became frustrated by the restrictions that software licenses placed on the use and modification of computer programs.[174] Stallman felt that these restrictions inhibited the effective and efficient development of new and innovative software technology by sticking a wrench in the collaborative programming process.[175] Dennis Kennedy explains that "Stallman and others believed

---

173. *See, e.g.*, Lerner & Tirole, *supra* note 56, at 22; John C. Yates & Paul H. Arne, *Open Source Software Licenses: Perspectives of the End User and the Software Developer*, 823 PRACTISING L. INST. 97, 104 (2005); Ieuan G. Mahoney & Edward J. Naughton, *Open Source Software Monetized: Out of the Bazaar and into Big Business*, 21 NO. 10 COMPUTER & INTERNET L. 1, 1 (Oct. 2004).

174. Kennedy, *supra* note 11, at 349. One author uncovered a story told by Stallman about the first laser printers at MIT's Artificial Intelligence lab that exemplifies Stallman's frustration:

> The laser printers of the mid-1970's were the size of today's compact cars. When Xerox gave the AI lab a Xerox Graphics Printer, the only place for it was in the lab's ninth-floor machine room. Researchers connected the printer to the local area network that the lab was developing, and soon anybody in the building could print a 100-page document by typing in a few commands.
>
> That worked fine, except that sometimes the printer would run out of paper or jam, and dozens of other jobs would pile up. Other times there would simply be a lot of people wanting to print long documents, and the person who needed to print a single page would have to run up and down the stairs or babysit the printer until that page appeared. But since the programmers at the lab had the source code to the program that ran the printer, they could add features that solved these problems. Soon the printer was helping the lab run smoothly. "It would send you a message when your document had been printed," recalls Stallman. "It would send you a message if you had anything queued and there was a paper jam."
>
> All this changed in 1978, when Xerox replaced the machine with a new laser printer called a "Dover" but wouldn't share the printer's source code with the lab. "We wanted to put those features into the Dover program, but we couldn't," Stallman says. Xerox wouldn't put the features into the program either. "So we had to suffer with paper jams that nobody knew about."

Heffan, *supra* note 89, at 1504 n.101 (citing Simson L. Garfinkel, *Programs to the People*, TECH. REV., Feb.-Mar. 1991, at 53, 54).

175. Kennedy, *supra* note 11, at 349. The author notes:

> The programming culture of the time created an environment where programmers freely worked on programs with each other, contributed fixes for the general public good, and saw development in a community context where people were free to take advantage of the innovations and improvements that others created, while still giving attribution and acknowledgement for the efforts of individual programmers.

Id.

that proprietary, commercial development of software would lead to a number of problems relating to security, loss of innovation, incompatibilities and the like, in part because it reduced the number of skilled, independent programmers who could analyze and correct source code."[176]

With the assistance of law professor Eben Moglen, Stallman drafted the first General Public License (GPL) and began the work of developing software to be distributed under the license.[177] The GPL is considered by many to be the most restrictive open source license because of its so-called "copyleft" provision. The copyleft provision ensures that any subsequent distribution of an open source project or derivative work is distributed under the GPL open source license. This has been pejoratively described as the "viral aspect" of the GPL. This aspect will be more fully discussed in subsequent sections.

In 1985, Stallman founded the Free Software Foundation (FSF) to encourage development of open source software and distribution under the GPL.[178] The superstar of the open source movement, Linux, is an operating system distributed under the GPL. Linux has recently gained popular attention by nibbling away at Microsoft's

---

176. *Id*. at 349-50. Stallman's concerns again raise that fundamental question embedded within the open source debate: Is collaboration more important than economic incentives in spurring innovative development of computer software? There is some inherent logic in the idea that many minds contributing to a project will be more effective, efficient, and creative than one or a few minds. This is the logic of the open source community. Beyond this utilitarian rationale, there is also an argument that there is inherent *a priori* value in sharing ideas for the good of the community. At a recent keynote symposium at the University of Maine School of Law, Professor Eben Moglen offered his view that incentives are no longer necessary to spur creative and innovative software:

> The Net is a superconductive medium for the creation of software. So, as I wrote in 1999, when it was a little less obvious than it is today, we are witnessing a phenomenon that was first noticed by Michael Faraday at the beginning of the nineteenth century. Wrap a coil around a magnet; spin the magnet. Electrical current flows in the wire. One does not ask, "What is the incentive for the electrons to leave home?" It is an inherent emergent property of the system, we have a name for it: we call it induction. The question we ask is, "What is the resistance of the wire?" Moglen's corollary to Faraday's Law says wrap the Internet around every brain on the planet; spin the planet. Software flows in the network. It is wrong to ask, "What is the incentive for people to create?" It is an emergent property of connected human minds that they do create.

Eben Moglen, *Freeing the Mind: Free Software and the Death of Proprietary Culture*, 56 ME. L. REV. 1, 4 (2004). Despite Moglen's creative effort to analogize human beings with magnets and coils, traditional economic logic suggests that incentives play an important role as well.

177. Kennedy, *supra* note 11, at 350. It should be noted that Stallman did not use the term "open source" software. Instead, Stallman coined the term "free software." Kennedy explains that "Stallman's vision and philosophy was that software should be free (as in speech, not as in beer) . . . ." *Id*. at 349. There is continuing disagreement as to the appropriate use of each term.

178. 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, 183 (1999). *See also* Krysten Crawford, *Linux Litigation Endangers Future of the Open-Source Software Movement*, BROWARD DAILY BUS. REV., Mar. 17, 2004, at 8. Crawford reports:

> According to industry analysts, Linux is poised to become the No. 2 computer operating system in 2004, after Microsoft Corp.'s Windows . . . . Right now there are an estimated 2.6 million Web and file servers running Linux. Computer research company Gartner Inc., estimates that the Linux server market, worth $2.6 billion now, will become a $6.3 billion business by 2008.

Id.

monopolistic share of the PC operating system market and already has a substantial presence in the server market. Linux is leading the commercial mainstreaming trend of open source software that has occurred in recent years.

Contemporaneous with Stallman's efforts to establish the Free Software Foundation and promote the GPL, programmers at the University of California at Berkeley (Berkeley) were working on improvements to the Unix operating system.[179] The Berkeley efforts were similar to Stallman's free software projects in that programmers had free and open access to source code and "[p]rogrammers could make derivative works as they saw appropriate to fix bugs, make improvements and fine tune the program."[180] As the project became more popular, programmers drafted a license (the Berkeley Software Distribution License, hereinafter BSD license) that "allowed licensees to work with source code and make derivative works."[181] Although the BSD license was similar in many ways to Stallman's GPL, a crucial difference was that the BSD license did not require derivative works to be distributed under the same licensing terms.[182] In other words, the BSD license did not contain the now-controversial copyleft provision. The significance of this distinction will be discussed further in subsequent sections of this article. It is sufficient to note here that this difference signifies an early fault line within the open source community. The BSD license and the GPL represent the early days of the open source movement. These academic roots were nourished by both pragmatic and ethical concerns of software programmers.

B. Commercial Mainstreaming

More recently, open source software has stepped out of the computer labs of college campuses and offices of programming hobbyists, and into the mainstream commercial arena. The impact of the Internet upon the proliferation of open source projects cannot be understated. By reducing the transaction costs of collaboration, the Internet has facilitated widespread contribution from programmers around the globe.[183] Mahoney and Naughton discuss this trend toward commercial adoption of open source software in their article, *Open Source Software Monetized: Out of the Bazaar and into Big Business*.[184] With apparent despair they acknowledge that:

> [O]pen source software is big business. Some of the largest owners of proprietary computer technology, companies like Sun Microsystems, Hewlett Packard, and even IBM—the very icon of the industry that Stallman rejected—have embraced open

---

179. Kennedy, *supra* note 11, at 351.

180. *Id*. at 352.

181. *Id*.

182. *Id*.

183. One author explains:

> Contributors to an open source program are often geographically dispersed . . . . Use of modern communication technologies—project web pages, mailing lists, newsgroups, et cetera—are necessary to facilitate the desired peer feedback that is at the heart of the open source process. In short, the growth of the Internet and Internet technologies has made the open source method possible.

Marcus Maher, *Open Source Software: The Success of an Alternative Intellectual Property Incentive Paradigm*, 10 FORDHAM INTELL. PROP. MEDIA & ENT. L.J. 619, 626 (2000) (citations omitted).

184. Mahoney & Naughton, *supra* note 173, at 1.

> source software. These companies have seized open source software as a strategic weapon, one that now represents a significant and growing source of revenue for their shareholders.
>
> The open source movement may have been born as a political ideology, a communitarian alternative to corporate profit seeking and the 'privatization' of technical innovation, but it has been transformed into a commercial enterprise.[185]

1. *Netscape*

In 1998, Netscape surprised many by releasing its popular Navigator Internet browser under an open source license.[186] At least one author suggests that Netscape was influenced at the time by an essay espousing the practical benefits of open source software development.[187] In his essay, *The Cathedral and the Bazaar*, programmer Eric Raymond argues that open source software development (the Bazaar) produced higher quality software more efficiently than proprietary software development (the Cathedral).[188] Decision-makers at Netscape were intrigued with the idea of developing a higher quality browser software product at lower costs to compete with Microsoft's Internet Explorer. One Netscape executive articulated what he perceived to be the competitive advantage of adopting the open source approach by saying that "[f]or Netscape, this gives us a way to engage the creative, innovative abilities of literally orders of magnitude more people than we could ever—really any commercial software company could ever afford to just put on their payroll."[189] Since Netscape's bold adoption of the open source approach in 1998, commercial interest in open source software has steadily increased, with Linux leading the way.

2. *Linux*

Undoubtedly, Linux is the poster child of the open source movement and has garnered much of the commercial investment in the open source approach.[190] Linux is an open source operating system. Operating systems are themselves software programs, but they are mission critical software programs because they control the computer hardware.[191] Software applications require an operating system as a platform to communicate between the computer hardware and the specific application.[192]

In 1991, a young Finnish student named Linus Torvalds shared with the world his operating system "kernel." Torvalds was inspired by a seminal book entitled

---

185. *Id*.

186. Kennedy, *supra* note 11, at 353.

187. *Id*.

188. *See generally* Eric Raymond, *The Cathedral and the Bazaar*, *available at* http://www.firstmonday.dk/issues/issue3_3/raymond/index.html (last visited Oct. 13, 2005).

189. Nathan Newman, *The Origins and Future of Open Source Software*, *available at* http://www.netaction.org/opensrc/future/oss-whole.html (last visited Sept. 25, 2005) (citations omitted).

190. "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." Vetter, *supra* note 3, at 608.

191. *Id.* at 605.

192. *See id*. at n.113.

Operating System written by Andrew S. Tanenbaum.[193] Professor Tanenbaum had written an operating system called Minix and included the 12,000 lines of source code in his book.[194] In keeping with the collaborative roots of academic programmers and probably influenced by the efforts of Richard Stallman, Torvalds openly shared the source code of his kernel and eventually decided to license the project using the open source GPL.[195] Programmers across the globe became interested in Torvalds' kernel and integrated it with existing programs to create a complete functioning operating system—GNU/Linux (popularly known as Linux).[196] Linux is truly an unprecedented and amazing story. It is a privately provisioned public good.[197] It is a valuable resource that is both non-rivalrous and non-excludable; and it's free.[198]

As one measure of its success, IBM has essentially abandoned its business of programming its own version of Unix and has invested heavily in Linux.[199] In 2001, IBM pledged to invest $1 billion in Linux and by January of 2002, they claimed to have recouped nearly the entire investment.[200] At the same time that IBM, Hewlett Packard, Dell and other computer industry mainstays have embraced Linux and the open source approach, smaller companies, such as Red Hat, Caldera, and Debian have made Linux the foundation of their business model.[201] These companies compile software applications and distribute various Linux versions. They also charge for value-added services such as installation, consultation, and support.

Beyond the commercial success of Linux and Linux-based companies in the United States, the promise of Linux extends to the developing world. At a conference sponsored by the United Nations, a World Bank representative explained that "[developing] countries need cheap and efficient technology to make the giant leaps necessary to catch up with the rest of the world. Many are now using Linux, which looks to become the No. 1 operating system in China and India soon."[202] A growing list of governments is considering requirements or incentives that would encourage the use of open source software by government agencies.[203] Linux is just one

---

193. *See* Ragib Hasan, *History of Linux*, *Version 2.1*, (1999) https://netfiles.uiuc.edu/rhasan/linux/.

194. *Id*.

195. Vetter, *supra* note 3, at 607.

196. Maher, *supra* note 183, at 622. "In 1991, about ten people were using and modifying Mr. Torvald's original 10,000 lines of code; today, there are an estimated seven million people using Linux and the code has grown to approximately 1.5 million lines." Patrick K. Bobko, *Linux and General Public Licenses: Can Copyright Keep "Open Source" Software Free?*, 28 AIPLA Q.J. 81, 85 (Winter 2000) (citations omitted).

197. Vetter, *supra* note 3, at 607.

198. It is interesting to note that the open source approach is what makes Linux a non-excludable resource. Because it is licensed under the open source GPL license, it cannot be made into a proprietary, and therefore excludable, program.

199. Vetter, *supra* note 3, at 615.

200. Stephen Shankland, *IBM: Linux Investment Nearly Recouped*, CNET NEWS, Jan. 29, 2002, http://news.com.com/2100-1001-825723.html (last visited Oct. 13, 2005).

201. *See* Hasan, *supra* note 193.

202. Michelle Delio, *Developing World Needs Linux*, WIRED NEWS, http://www.wired.com/news/politics/0,1283,59334,00.html? (last visited Oct. 6, 2005).

203. *See, e.g.,* Todd Benson, *Brazil: Free Software's Biggest and Best Friend*, N.Y. TIMES, March 29, 2005, at C1 ("Looking to save millions of dollars in royalties and licensing fees, Mr. da Silva has instructed government ministries and state-run companies to gradually switch from costly operating systems made by Microsoft and others to free operating systems, like Linux.").

example—albeit the best example—of the promise and potential of open source software.

3. *Apache*

Linux, however, is by no means the only successful open source project. The Apache Web server—an open source project—delivers the majority of Web pages viewed by Internet users around the world on the World Wide Web.[204] Web servers are fundamental software technology for the Internet. "Products like Apache make the Web the Web, just as operating systems make a computer a computer."[205] When a user clicks on a link, the Web server sends information from that server computer to the user's computer.[206] In a recent survey, the Apache web server was used by more than 67% of Internet websites, compared with Microsoft's 21% market share.[207] In the mid-1990s a group of Webmasters—connected by the Internet—formed the Apache Software Foundation (ASF).[208] The ASF utilized freely available source code from an early Web server that had been developed by the National Center for Supercomputer Applications (NCSA).[209] After modifying and adding to the original source code, the ASF released a new Web server under a specialized Apache open source license.[210] The Apache license is very close to a complete contribution to the public domain. It merely requires that certain attributions and notices be passed to downstream users as a condition of use.[211] Marcus Maher explains that "[t]he advantages of Apache include the fact that it is free (as in costless), that it is free (as in open source), and that it provides high quality performance."[212]

4. *Microsoft Responds*

The tripartite benefits of Apache, Linux, and other open source products have forced proprietary software developers to think carefully about how to respond to increasing commercial interest in the open source approach.[213] Microsoft's concern with the growth of Linux and other open source projects was exhibited as early as October of 1998.[214] An internal Microsoft memorandum from 1998 revealed the concern that Microsoft had with regard to the growth of Linux and other open source projects.[215]

---

204. Vetter, *supra* note 3, at 605.

205. *Id.* at 609 n.132.

206. *Id*.

207. This survey was conducted by the Netcraft Web Server Survey, http://news.netcraft.com/archives/web_server_survey.html (last visited Sept. 25, 2005).

208. Vetter, *supra* note 3, at 610.

209. *Id*.

210. *Id*.

211. *Id*. at 611.

212. Maher, *supra* note 183, at 621.

213. Other well-known open source software products include the Perl programming language, Sendmail (an Internet e-mail router), Jboss application server, MySQL database, and the Mozilla open source Web-browser. *Id.* at 623-25.

214. Stephen R. Walli, *Perspectives on the Shared Source Initiative*, Mar. 24, 2005, http://www.onlamp.com/pub/a/onlamp/2005/03/24/shared_source.html.

215. The memo states:

In 2001, Microsoft announced that it would begin sharing source code to some of its software programs with limited audiences as part of its new Shared Source Initiative.[216] Under the program, Microsoft customers and independent developers are permitted to download the source code of a limited number of Microsoft programs, examine it, and copy it for personal use.[217] Academic researchers are given additional permission to make limited modifications to the source code.[218] However, software developers are not permitted to modify the code or distribute copies without a Microsoft license.[219] Open source proponents have responded critically and skeptically to the Shared Source Initiative; nonetheless, Microsoft's active response to the open source movement and its recognition of the benefits of sharing source code are worth noting. This is further evidence that the open source approach is sustainable and is here to stay.

C. Making Sense of Open Source Success

The commercial success of Linux, Apache, and other open source projects has occurred despite the fact that there are substantial legal uncertainties about the validity of open source licenses in general and specific license provisions in particular. One can only speculate whether this momentum can continue in the face of these uncertainties or whether resolution of some of these questions—judicially or legislatively—will be necessary for open source projects to continue to prosper. A uniform code governing software transactions, such as UCITA, could help to create more certainty and legitimacy with regard to open source licenses, thereby promoting the open source movement. However, some have suggested that UCITA will have a detrimental impact on open source development. Understanding why the open source approach has been so successful might inform an analysis of whether UCITA will benefit or harm the open source movement.

A fair amount of academic scholarship has been dedicated to the task of explaining the relative success of the open source movement. Specifically, scholars have offered various theories about why individual programmers choose to volunteer their time to open source projects, as well as why some open source projects have been embraced by for-profit commercial enterprises. Marcus Maher uses the science of complexity

---

> Open Source Software (OSS) is a development process which promotes rapid creation and deployment of incremental features and bug fixes in an existing code/knowledge base. In recent years, corresponding to the growth of Internet, OSS projects have acquired the depth & complexity traditionally associated with commercial projects such as Operating Systems and mission critical servers.
>
> Consequently, OSS poses a direct, short-term revenue and platform threat to Microsoft —particularly in server space. Additionally, the intrinsic parallelism and free idea exchange in OSS has benefits that are not replicable with our current licensing model and therefore present a long term developer mindshare threat.

Vinod Valloppillil, *Microsoft Memo: Open Source Software: A (New?) Development Methodology*, Aug. 11, 1998, http://www.scripting.com/misc/halloweenMemo.html.

216. *See* Walli, *supra* note 214.

217. *Microsoft Depends On Shared Source, Dips Toe In Open-Source Waters*, TECHWEB NEWS, Apr. 7, 2005, *available at* 2005 WLNR 5493721.

218. *Id.*

219. *Id.*

theory to explain the development and success of the open source movement.[220] He explains that the open source methodology is a complex system and that "complexity theory can explain the observed result of the technical strength of open source projects."[221] In laying the foundation for this analysis, Maher suggests that the success of the open source movement cannot be adequately understood in terms of the economic incentives system created by federal intellectual property law.[222] He first argues that "[p]roduct comparisons show that open source software attains high technical standards despite the relative absence of economic motivation for the creators of this software,"[223] and provides several examples of the most successful open source projects.[224] Maher also proposes various theories to explain why individual programmers and firms choose to contribute their time and effort to open source projects, despite the fact that there is no direct economic reward. He first offers Eric Raymond's "gift culture" theory: "[I]n gift cultures, social status is determined not by what you control but by what you give away."[225] Programmers receive recognition and status from contributing code to open source projects. Programmers that contribute good source code to projects gain prestige among project contributors and possibly the larger open source community.[226] Another reason offered for contribution to the open source approach is the ability to control and modify the code to meet specific needs of the firm or the programmer.[227] This flexibility of open source software offers a valuable advantage over proprietary software.[228] Finally, Maher suggests that "[t]he pure pleasure of hacking is another reason why individuals contribute to open source projects."[229] The open source community provides professional programmers with an alternative arena to write more creative code and participate in projects that would be unavailable to them during their day job. All of these factors probably play a role in creating the necessary incentives for programmers to contribute to open source projects. All this suggests that there are additional reasons for programmers to contribute to open source projects beyond the economic incentives provided by intellectual

---

220. Maher, *supra* note 183, at 658.

221. *Id*.

222. "The open source software movement, which has gained publicity as the popularity of the Linux operating system has grown, provides an alternative to the economic incentives that dominate the thinking of U.S. intellectual property policy." *Id*. at 619.

223. *Id*.

224. These include the Apache web server, the Berkeley Internet Name Domain package (BIND), the Linux operating system, the Mozilla web browser, the Perl programming language, and the Sendmail internet e-mail router. *Id*. at 621-24.

225. *Id*. at 632.

226. *Id*. Maher suggests:

> Prestige within the open source community may allow a programmer to more readily persuade others to join projects owned by such a person, or to place particular value on that person's input. To a lesser extent, the prestige of a developer in the open source community may also spill over to the exchange economy, placing them in higher demand within that market.

Id.

227. *See id*. at 633.

228. For a concrete example of the benefits of flexibility, see *supra* note 175.

229. Maher, *supra* note 183, at 635.

property protections. Maher's use of complexity theory to analyze the open source movement is a novel and creative approach.

Complexity theory has been used in various fields (e.g., genetics and economics) to explain behaviors and outcomes that cannot be adequately understood using traditional theories.[230] In this respect, it is well suited for analyzing the surprising growth of the open source movement and the superior technical merit of many open source projects. "[C]omplex systems are groups of agents whose nature and behavior are governed by certain sets of rules. The nature and behavior of these agents lead to outcomes within the system and capabilities of the system making it greater than the sum of its parts."[231] Leaving the details of complexity theory for those who would choose to read Maher's article, open source methodology is a complex system because many individual programmers (agents) contribute in semi-structured ways to open source projects. Maher states that the "main consequences of complexity are adaptability and emergence."[232] The interaction of many contributing programmers and the flow of information and source code through the open source system give rise to unpredictable yet impressive outcomes. To a complexity theorist, the technical sophistication of open source software is not surprising; rather, it is precisely what is expected from a complex system. Maher analyzes the development of the Linux operating system in terms of complexity theory as a concrete and specific example of the development and fruition of a complex open source project.[233] Recognizing that complex systems are highly unpredictable, Maher makes no effort to predict the future of the open source movement. He does, however, offer some of the potential risks for the open source movement to avoid. Specifically, he suggests that the open source community should avoid driving the open source approach away from complexity and toward chaos in one direction and linearity in the other.[234]

Of particular relevance for this paper, Maher is concerned about the validity of some open source licenses. He recommends the adoption of a uniform code—UCITA—to ensure that open source license provisions such as copyleft and disclaimer of implied warranties are enforced.[235] According to Maher, "the UCITA, if adopted by the states, could help resolve some of the uncertainty regarding open source licenses."[236] Some of these legal uncertainties will be discussed below.

D. Open Source Licenses

Although the open source movement might still be described as being in its infant stages, there is substantial academic scholarship addressing various aspects of the topic. A survey of the relevant legal scholarship reveals that open source licensing has been the prominent focus of many, if not most, contributions.[237] This makes sense

---

230. *Id*. at 646.
231. *Id*.
232. *Id*. at 655.
233. *See id*. at 664.
234. *Id*. at 669.
235. *Id*. at 677-78.
236. *Id*. at 678.
237. *See, e.g.*, Kennedy, *supra* note 11; Gomulkiewicz, *supra* note 178; Natasha T. Horne, Comment, *Open Source Software Licensing: Using Copyright Law to Encourage Free Use*, 17 GA. ST. U. L. REV. 863 (2001); Nadan, *supra* note 7.

given that licensing is the primary mode of distribution for software—open source and proprietary alike—as opposed to outright sale or lease.[238]

A myriad of open source licenses have been promulgated and used for distributing open source software.[239] Much of the academic scholarship has attempted to catalog and classify the most prominent open source licenses. In the spirit of the collaborative open source approach, I will forego an independent classification of my own and rely instead on a "taxonomy of licenses" formulated by Lawrence Rosen.[240] Rosen classifies open source licenses into four categories: Academic Licenses, Reciprocal Licenses, Standards Licenses, and Content Licenses.[241] The BSD license, introduced above, is an academic license and is probably the least restrictive open source license. The GPL, also discussed above, is a reciprocal license and is much more restrictive. Although these two licenses share in the early academic history of the open source approach, they also represent the genesis of two separate factions of the open source community. Richard Stallman, author of the GPL, argues that the BSD license is anathema to the ideals of the original "free software" community. Stallman, and his compatriots at the FSF, might be described as software moralists. They believe that restricting access to source code is both morally wrong and inefficient.[242] The BSD license permits such restrictions by allowing licensees that create derivative works to distribute their work under any license, including proprietary licenses. Richard Stallman explains the difference between his free software movement and the open source approach this way:

> Free software and open source are the slogans of two different movements with different philosophies. In the free software movement, our goal is to be free to share and cooperate. We say that non-free software is antisocial because it tramples the users' freedom, and we develop free software to escape from that.

---

238. Quittmeyer, *supra* note 93, at 909 ("Typically, vendors of software programs transact business by 'licensing' rather than 'selling' software.").

239. At present there are nearly sixty licenses approved by the Open Source Initiative (OSI) and listed on their website. These licenses are available at http://www.opensource.org/licenses (last visited Sept. 16, 2005). OSI is one of several organizations dedicated to the promotion of open source software. *See infra* notes 273-79 and accompanying text for a discussion of OSI and the Open Source Definition.

240. *See* ROSEN, *supra* note 105, at 51.

241. *Id.* at 69-71.

> *Academic licenses,* so named because such licenses were originally created by academic institutions to distribute their software to the public, allow the software to be used for any purpose whatsoever with no obligation on the part of the licensee to distribute the source code of derivative works. . . . Academic licenses create a public commons of free software, and anyone can take such software for any purpose—including for creating proprietary collective and derivative works—without having to add anything back to that commons.

Id. at 69-70. "*Reciprocal licenses* also allow software to be used for any purpose whatsoever, but they require the distributors of derivative works to distribute those works under the same license, including the requirement that the source code of those derivative works be published." *Id.* at 70. "*Standards licenses* are designed primarily for ensuring that industry standard software and documentation be available to all for implementation of standard products." *Id.* "*Content licenses* ensure that copyrightable subject matter other than software, such as music, art, film, literary works, and the like, be available to all for any purpose whatsoever." *Id.* at 71.

242. "Free software is a matter of freedom: people should be free to use software in all the ways that are socially useful." Philosophy of the GNU Project, http://www.gnu.org/philosophy/philosophy.html#AboutFreeSoftware (last visited Sept. 22, 2005).

> The open source movement promotes what they consider a technically superior development model that usually gives technically superior results. The values they cite are the same ones Microsoft appeals to: narrowly practical values.
>
> Free software and open source are also both criteria for software licenses. These criteria are written in very different ways but the licenses accepted are almost the same. The main difference is the difference in philosophy. Why does the philosophy matter?
>
> Because people who don't value their freedom will lose it. If you give people freedom but don't teach them to value it, they won't hold on to it for long. So it is not enough to spread free software. We have to teach people to demand freedom, to fight for freedom. Then we may be able to overcome the problems that today I see no way to solve.[243]

Stallman's GPL achieves this freedom by requiring all licensees to distribute GPL licensed code, or any derivative work, under the GPL. Specifically, section 2(b) states that "[y]ou *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*."[244] Rosen describes this provision as reciprocal because it imposes an obligation upon the licensee that chooses to distribute GPL code or any derivative program.[245] Section 2(b) is also described as the "copyleft provision" and pejoratively as the "viral provision." Open source advocates call this provision "copyleft" because it uses copyright law to ensure that source code remains open for downstream users.[246] Open source opponents call this provision "viral" because it can potentially infect a software development project that incorporates GPL licensed source code.[247] This is a potential pitfall awaiting programmers that want to take their software project private and license it under a proprietary license.[248] As commercial software developers consider utilizing open source software, this aspect of the GPL license is a serious concern. In the absence of

---

243. Interview: Richard Stallman, http://kerneltrap.org/node/4484 (last visited Sept. 22, 2005).

244. GNU General Public License, Version 2, Section 2(b) (1991) http://www.gnu.org/licenses/gpl.html (last visited Apr. 16, 2005) (emphasis added).

245. ROSEN, *supra* note 105, at 70 ("The GPL license, written by Richard Stallman and Eben Moglen at the Free Software Foundation, is the archetypal reciprocal license. Anyone who creates and distributes a derivative work of a work licensed under a reciprocal license must, in turn, license that derivative work under the same license.").

246. One commentator explains:

> [C]opyleft licenses provide that you may distribute the open source code and any modifications to it, but only under the same open source license under which you received it. In this way, as the code and modifications to it pass from person to person or entity to entity, they remain open source. This ingenious tactic is embodied in the GPL, but not the BSD license.

Nadan, *supra* note 7, at 357-58.

247. *See* Vetter, *supra* note 3, at 633-34.

248. One author warns that "if you incorporate some GPL code in your proprietary software product, arguably your whole proprietary product becomes open source and must be licensed by you under the GPL. This illustrates the importance of fully understanding open source licensing before deploying it in a commercial context." Nadan, *supra* note 7, at 359. Nadan concludes that the GPL license "must be used very cautiously" and suggests that for commercial developers the BSD license may be a better approach. *Id.* at 361.

any judicial guidance, the literature is rife with ruminations about the appropriate scope and enforceability of this provision.[249]

VII. LEGAL UNCERTAINTIES

Perhaps the most glaring aspect of open source licensing is the near complete dearth of litigation concerning the validity of open source licenses. In 2001, one open source commentator wrote that "[i]t is important to note that no cases have been decided that directly interpret any of the Open Source licenses or the particular issues arising under specific licenses . . . . In other words, there are only questions with regard to the legal issues involving Open Source licenses, not any definitive answers." [250] With some minor exceptions, the legal landscape of open source software has not changed.[251] Practitioners and academics in the open source arena are currently operating in a kind of legal vacuum, devoid of any judicial guidance concerning the legitimacy of open source licenses. Eben Moglen, co-author of the GPL, has emphasized recently that the Free Software Foundation has been able to enforce the GPL without resorting to litigation.[252]

Despite Moglen's comments, the GPL is not likely to remain unchallenged for long. Unless and until courts have this opportunity to weigh in, or states resolve the issue by adopting a uniform code, questions will remain. While the recent commercial success of Linux and other open source projects suggest that this lack of certainty has not inhibited progress, one can only speculate how many interested parties are waiting on the sidelines until some of the questions are resolved.

It is worth noting that a recent lawsuit has the potential to change this barren legal landscape. In recent months there has been much discussion about the pending litigation between the SCO Group (SCO) and IBM and the potential impact this case may have on the open source movement.[253] After a failed collaborative software development arrangement between the two companies, SCO claimed that IBM misappropriated proprietary Unix source code by including parts of the code in IBM's version of the

---

249. *See, e.g.*, Joseph Scott Miller, *Allchin's Folly: Exploding Some Myths About Open Source Software*, 20 CARDOZO ARTS & ENT. L.J. 491 (2002); Nadan, *supra* note 7, at 369 (arguing that the GPL may constitute a misuse of copyright); Jason B. Wacha, *Taking the Case: Is the GPL Enforceable?*, 21 SANTA CLARA COMPUTER & HIGH TECH. L.J. 451 (2005) (rebutting various arguments that the GPL is unenforceable).

250. Kennedy, *supra* note 11, at 368.

251. The GPL has been raised as a secondary issue in a few cases in the U.S., but courts have not yet been required to rule on its enforceability. *See* Wacha, *supra* note 249, at 454.

252. Moglen explains:

> For ten years, I did all of the GPL enforcement work around the world by myself, while teaching full time at a law school. It was not hard, really; the defendant in court would have had no license, or had to choose affirmatively to plead my license: they did not choose that route. Indeed they did not choose to go to court; they cooperated, that was the better way. My client did not want damages, my client wanted compliance. My client did not want publicity, my client wanted compliance. We settled for compliance all the time. We got compliance all the time.

Moglen, *supra* note 176, at 7.

253. *See* Kerry D. Goettsch, *SCO Group v. IBM: The Future of Open-Source Software*, 2003 U. ILL. J. L. TECH. & POL'Y 581.

Linux operating system.[254] In addition to its legal claims against IBM, SCO has also sent notice to scores of IBM's Linux customers threatening copyright infringement suits unless they pay licensing fees to SCO.[255] At present it remains unclear whether this lawsuit will implicate the GPL license under which Linux is distributed. One author suggests:

> [C]ourts could potentially invalidate the GPL, regardless of the ultimate outcome of the suit against IBM, since SCO's case is not wholly dependent on the issue. This would have an enormous effect on the development of all types of open-source software. At the same time, the courts may dodge the issue entirely or even affirmatively validate the GPL, both of which could be seen as positive outcomes for the open-source community.[256]

Any judicial guidance regarding the enforceability of the Linux GPL license would be helpful. Yet, this would not resolve the issue of different interpretations among different courts. This dispute exemplifies the potential chilling effect that legal uncertainty can have on the open source movement.

Much of the legal uncertainty regarding open source software relates to the basic question of whether open source licenses are enforceable contracts or merely bare licenses. To the extent that open source licenses are deemed to be enforceable contracts, licensors will benefit from stronger enforcement of specific license terms and provisions. As noted above, however, a license is not necessarily a contract. A bare license is merely permission authorizing others to exercise rights that would otherwise be prohibited by law.[257] For example, a driver's license authorizes the licensee to drive on public roads, which would otherwise be prohibited by law.[258] Unlike a contract, this type of bare license does not create any bilateral obligation between the two parties. In the context of software, the author of a software program owns the copyright in that program and this property right is recognized and protected by the Copyright Act. As such, the author enjoys certain exclusive rights such as the right to makes copies of the program and the right to distribute those copies.

As an example, consider a programmer, Anne, who has written a useful software program. Anne wants to allow friends to use the program for personal use, but restrict them from distributing copies to others. Anne could give her friends limited permission to make copies for themselves only, but not grant permission to distribute. If Anne's friend, Donna, were to make and use a personal copy, that would be permitted by the license. However, if Donna made a copy and sent it to a friend, Chris, then Donna would be exceeding the scope of her license and would be liable for infringing on Anne's exclusive distribution right. This same arrangement could also be structured as a contract if the parties wished. Anne could offer to Donna a permission to make a personal copy of the program in exchange for $10. If Donna accepts the offer, and pays Anne the $10, a contract is formed. The only difference here is the offer, acceptance, and the $10 consideration. This minor difference does,

---

254. *Id*. at 583-84.
255. *Id*. at 584-85.
256. *Id*. at 587.
257. ROSEN, *supra* note 105, at 53.
258. *Id*. at 51.

however, alter the arrangement significantly. First, the contractual arrangement provides an additional cause of action for Anne, if Donna subsequently sends an additional copy to Chris. Anne could sue Donna for copyright infringement as above, but she could also sue for breach of contract, provided the terms of the agreement are clear enough. The contract also protects Donna by making the copying permission irrevocable and creating a cause of action against Anne in the case of a breach.

One prominent open source advocate, Lawrence Rosen, suggests that "it is safer for a licensor and his licensees to enter into enforceable contracts."[259] Presumably, it is safer because both parties are mutually obligated by the terms of the contract. This reduces the risk of unanticipated revocation by the licensor and ideally creates mutual expectations about the arrangement. Rosen also points out that under contract law, there are rules and procedures for interpreting vague or ambiguous terms in an agreement and for filling in gaps where the agreement is silent.[260] This fosters uniformity and certainty with respect to license interpretation. Despite Rosen's preference for contract, the FSF has remained adamant that their license, the GPL, is a bare license, not a contract.[261] The GPL, however, is not just a mere permission. It imposes obligations upon licensees that must be accepted in order to exercise the rights granted in the license.[262] The most obvious example of such an obligation is the copyleft requirement to license any derivative work under the GPL. Given these obligations, many have suggested that the GPL would likely be interpreted as a contract, not a bare license.[263]

If the GPL and other open source licenses are considered contracts, as many commentators believe, they must meet specific requirements to be enforceable. Under contractual analysis, there must be an offer, an acceptance, and consideration for the license to be an enforceable contract.[264] Several commentators have suggested that the licenses themselves may be completely unenforceable based on traditional contract doctrine of offer and acceptance. Similar to the debate about shrinkwrap and clickwrap licensing agreements, it is questionable whether open source licenses involve the kind of acceptance that is required under contract law. One author suggests that when compared with traditional shrinkwrap and clickwrap agreements, some open source licenses are less likely to be enforced because of a lack of notice of license terms.[265]

---

259. *Id*. at 56.

260. *Id*. at 57-58.

261. Wacha, *supra* note 249, at 456. Eben Moglen of the FSF has explained that "[t]he GPL, however, is a true copyright license: a unilateral permission, in which no obligations are reciprocally required by the licensor." *The GPL is a License, Not a Contract, Which is Why the Sky Isn't Falling*, GROKLAW, December 14, 2003, http://www.groklaw.net/article.php?story=20031214210634851.

262. Wacha, *supra* note 249, at 456.

263. *Id*. One wonders why the FSF would prefer to call the GPL a bare license instead of a contract. It is quite possible that they want to avoid any warranties and liabilities that would be implied into the contract by application of the UCC or, potentially, UCITA. This, however, is mere speculation on my part.

264. ROSEN, *supra* note 105, at 59.

265. Nadan explains:

> Rather than requiring acceptance of a clickwrap agreement for downloads from a website, acceptance of a clickwrap license that pops up during installation of the code, or acceptance of a shrinkwrap license included in the packaging, the open source licenses just require that a notice about the license be provided in or with the code. No manifestation of assent is required to prove the licensee agreed to the terms, and indeed the user can access the code

If the licensee or user has no notice of the licensing agreement, they cannot be contractually bound by the agreement. This might become a problem if, for example, Donna creates a derivative work based on a GPL licensed program written by Anne. If Donna obtained the source code without notice of the GPL terms and then decided to license the derivative work to others using a proprietary license, in direct violation of the copyleft provision, then Anne may have no cause of action for breach of contract. The FSF would likely respond that this, however, does not foreclose the possibility of a copyright infringement claim; and they are probably right. The copyright exists regardless of whether a contract has been formed between the licensee and licensor. Nevertheless, for open source licensors that want the support of contract law, they must ensure that the transaction includes the required offer, acceptance, and consideration. Nadan suggests that "[o]ne initial solution is to set up the code download site so that the user is forced to click 'I accept' to a clickwrap form of the GPL before downloading."[266] This would ensure that notice of the GPL terms are provided before the source code can be obtained. As mentioned above, to the extent that these types of agreements are enforceable as contracts, this would bolster the likelihood of open source license enforcement. Yet, even the enforceability of clickwrap agreements is not free from all doubt.

Even if open source licenses in general are considered to be enforceable contracts, some specific open source provisions may still be held unenforceable on a number of different grounds. Several commentators have argued that the copyleft provision of the GPL in particular is one such provision and might be void as a misuse of copyright.[267] Copyright misuse is an equitable defense to a copyright infringement action. A finding of copyright misuse would preclude enforcement of the copyright where the copyright holder is engaging in conduct—usually by contract—that expands the scope of protection beyond the statutory rights granted in the Copyright Act.[268] Christian Nadan argues that:

> The copyleft provision purports to infect independent, separate works that are not derivative of the open source code, and requires that such independent works be licensed back to the licensor and the entire world under the GPL. The Copyright Act does not give the copyright owner rights to such independent nonderivative works. Attempting to extract such rights exceeds the scope of the copyright.[269]

The idea that the copyleft provision might be considered a misuse was certainly not beyond the consideration of Moglen and Stallman. Accordingly, they include the

---

> without ever seeing the license, let alone agreeing to it. This greatly increases the risk that no license agreement has been formed.

Nadan, *supra* note 7, at 362.

266. *Id*. at 367. In general, courts have been willing to enforce these clickwrap agreements, but have not agreed on a consistent mode of analysis. *See supra* Part IV.

267. *See, e.g*., Nadan, *supra* note 7, at 367.

268. *See id*. Courts first applied copyright misuse by analogy to the judicially recognized defense of patent misuse. Nadan provides, as an example of patent misuse, "a contract that requires a patent licensee to continue to pay royalties beyond the statutory term of the patent. A patent generally lasts for 20 years from its filing date. Demanding royalties after the patent expired would exceed the scope of the rights granted in a patent." *Id*. at 367-68.

269. *Id*. at 369.

following disclaimer in the GPL: "[I]t 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."[270] Whether this disclaimer shields a copyright misuse defense remains unclear and would probably require a relatively fact-specific analysis. The point here is that even if open source licenses are generally enforceable, some specific provisions therein may not be. Moreover, even if the copyleft provision is found to be an enforceable license term, there is still uncertainty about the extent to which it can bind downstream users of GPL licensed code.

At present, many within the open source community are also concerned with the proliferation of licenses.[271] As previously mentioned, there are currently nearly sixty separate licenses approved by the Open Source Initiative (OSI).[272] The OSI is a self-appointed standards organization that has attempted to offer some organization to the inherently decentralized open source approach. The OSI evaluates and certifies licenses for compliance with their Open Source Definition.[273] Despite the large and

---

270. GNU General Public License, Version 2, Section 2 (1991), http://www.gnu.org/licenses/gpl.html.

271. "[T]he Linux and open source software industry is facing some of its biggest challenges in containing the proliferation of its own licenses . . . ." Jay Lyman, *License issues lining up for Linux*, *open source*, NEWSFORGE, Feb. 4, 2005, http://software.newsforge.com/article.pl?sid=05/02/03/206216.

272. *See supra* note 239.

273. The Open Source Definition provides that:

> Open source doesn't just mean access to the source code. The distribution terms of open-source software must comply with the following criteria:
>
> 1. Free Redistribution
>
> The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.
>
> 2. Source Code
>
> The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost–preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.
>
> 3. Derived Works
>
> 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.
>
> 4. Integrity of The Author's Source Code
>
> The license may restrict source-code from being distributed in modified form *only* if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.
>
> 5. No Discrimination Against Persons or Groups
>
> The license must not discriminate against any person or group of persons.
>
> 6. No Discrimination Against Fields of Endeavor
>
> The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.
>
> 7. Distribution of License

growing number of OSI certified licenses, the vast majority of open source projects rely on either the BSD license or the GPL.[274] Although license proliferation is one indicia of the success of the open source movement, it does present some problems. First and foremost is the problem of license incompatibility. The propagation of many different and incompatible licenses frustrates programmers that want to share their source code with each other.[275] This would negate one of the primary benefits of the open source approach: the ability to recycle existing source code into new open source projects. Several authors have been consistently critical of this state of affairs and warned that disorganization could lead to problems. In addressing the issue of open source license proliferation, one author commented that "[t]here are a multitude of licenses that purport to meet the goals of open source development. These licenses reflect different, and sometimes contradictory, approaches to core licensing issues. Many of the licenses are buggy—out of date, misapplied, misunderstood and hopelessly confusing. This state of affairs benefits no one."[276] That same author recommends a standards organization to bring certainty and uniformity to the open source licensing regime, but is skeptical of the OSI's capability in this regard.[277] The OSI, nevertheless, recognizes that license proliferation is a concern and has recently taken steps to remedy the problem. The OSI is currently considering proposals to add three new criteria to the evaluation and certification process that should limit the number of

---

> The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.
>
> 8. License Must Not Be Specific to a Product
>
> The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.
>
> 9. License Must Not Restrict Other Software
>
> The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.
>
> 10. License Must Be Technology-Neutral
>
> No provision of the license may be predicated on any individual technology or style of interface.

Open Source Initiative, http://www.opensource.org/docs/definition.php (last visited Sept. 20, 2005) ("rationale" sections omitted).

274. "While there are more than fifty OSI-approved licenses only a handful of those are actually in wide use; in fact, just two licenses (GPL and BSD) cover over 70% of all projects." Open Source Initiative, http://www.opensource.org/docs/policy/licenseproliferation.php (last visited Sept. 20, 2005).

275. *See* Stephen Shankland, *Open-source overseer proposes paring license list*, TECHREPULIC, Mar. 3, 2005, http://techrepublic.com.com/5100-22_11-5598102.html?part=rss&tag=feed&subj=tr# ("[Some experts] argue that having numerous licenses leads to numerous islands of incompatible open-source code. That means programmers can't share as much of their work. And, of course, it means those thinking of participating must spend more time understanding new licenses.").

276. Robert W. Gomulkiewicz, *De-Bugging Open Source Software Licensing*, 64 U. PITT. L. REV. 75, 76 (2002).

277. *Id.* at 77, 82.

additional licenses that will be certified.[278] This does not, however, solve the problem of the many open source licenses already being used.

A. UCITA and Open Source

The open source movement's relative success is perhaps even more impressive and surprising when one considers all of the legal questions that are yet to be resolved. To what extent would a uniform code such as UCITA promote or hinder the continuing development of open source licensing? There is certainly no unanimous response to this question, but for the most part, open source proponents were not in favor of UCITA. If, as most commentators believe, UCITA is applicable to open source licenses, one author suggests that "UCITA may imply terms contrary to the intent of the authors of a particular Open Source license. In particular, UCITA might imply terms relating to duration of the license, warranties and other restrictions."[279] When asked about his reaction to UCITA, Richard Stallman responded critically by saying that "UCITA would make it harder for us to avoid liability for bugs that turn up in free software we develop . . . . This is grossly unfair."[280] Stallman added that "UCITA would also give proprietary software developers a way to prohibit reverse engineering."[281] Others have echoed Stallman's claim that UCITA would have a detrimental effect on the open source approach. Shawn Potter suggests that "the UCITA is far from being an open source booster. Depending on the language state legislatures adopt, the UCITA can as easily invalidate open source click-wrap licenses. The UCITA also poses other threats to open source software development regarding warranties and reverse engineering."[282]

Interestingly, concerns about reverse engineering appear to be a little outdated. As indicated earlier, UCITA's proscription of reverse engineering was amended in 2002. Perhaps more significantly, UCITA was also amended to provide an exemption from certain implied warranties for "free software."[283] Thus, the new UCITA now appears to favor "free software" over proprietary software, at least with respect to

---

278. *See* Shankland, *supra* note 275 ("[T]he new proposed provisions would require that a license not duplicate existing licenses; that it be clearly written, simple and understandable; and that it be reusable by moving the names of specific individuals, projects or organizations into an accompanying attachment.").

279. Kennedy, *supra* note 11, at 369.

280. Robert Lemos, *Interview: GNU Guru Richard Stallman*, ZDNET NEWS, Mar. 11, 2000, http://news.zdnet.com/2100-9595_22-519022.html?legacy+zdnn.

281. *Id.*

282. Potter, *supra* note 6, ¶ 93.

283. Section 410 of UCITA now reads:

> NO IMPLIED WARRANTIES FOR FREE SOFTWARE.
>
> (a) [Free software defined.] In this section, "free software" means a computer program with respect to which the licensor does not intend to make a profit from the distribution of the copy of the program and does not act generally for commercial gain derived from controlling use of the program or making, modifying, or redistributing copies of the program.
>
> (b) [Implied warranties inapplicable.] The warranties under Sections 401 and 403 do not apply to free software.

Uniform Computer Information Transactions Act § 410 (amended 2002), *available at* http://www.law.upenn.edu/bll/ulc/ucita/2002final.htm. Section 401 covers implied warranties for noninfringement and noninterference. *Id.* § 401. Section 403 provides an implied warranty of merchantability. *Id.* § 403.

implied warranties. These developments would seem to call for a reevaluation of UCITA from the open source community. Additionally, even if UCITA itself is dead in the water, the exemption from implied warranties for free software indicates that the open source approach is now on the radar of the NCCUSL.

Concededly, UCITA is not a magic pill. Nevertheless, there is little doubt that at least some uncertainties could be clarified by a uniform code, such as UCITA. Open source licenses are similar to shrinkwrap and clickwrap licenses in that they are standard form agreements that rely on the notice-plus-conduct model of contract formation discussed above. One author argues that the open source movement is dependent upon the validity of these "non-negotiated, standard-form, take-it-or-leave-it mass-market licenses."[284] Several authors insist that "UCITA, if adopted by the states, could help resolve some of the uncertainty regarding open source licenses."[285] These licenses—if enforced as contracts—allow software developers to disclaim implied warranties and shift the risk of liability for damages and infringement to the licensee. To the dismay of consumer advocates, UCITA would have validated these license terms and enforced them as contractual terms. Many critics viewed this treatment as unfairly deferential to the interests of large commercial software developers, such as Microsoft. This critique is by no means groundless. Microsoft would indeed benefit from contractual treatment of clickwrap and shrinkwrap license agreements. This was true in 1999, when UCITA was first offered to the states, and it is true today. But the commercial software landscape has been altered by the emergence of open source software. Now we must consider not only the benefits that UCITA would offer to Microsoft, but also the benefits it would offer to the open source movement. To the extent that UCITA would benefit the open source approach, it should be welcomed, regardless of its effect on Microsoft.

Enforcement of warranty and liability disclaimers was a particularly controversial aspect of UCITA. However, the emergence of the open source approach requires a rethinking of why these terms should be enforced. The ability of open source licensors to include these terms in their license agreements is crucial to the vitality of the open source movement.[286] Because of the collaborative nature of the open source development model, hundreds or even thousands of programmers contribute source code to open source projects. Liability for infringement and implied warranties of merchantability and fitness for a particular purpose are potentially substantial disincentives to contribution to open source projects.[287] Individual programmers that contribute on their own time for little direct reward would be foolish to risk exposing themselves to this liability. Obviously, the NCCUSL recognized this issue and responded by exempting distributors of free software from some of the implied warranties. At present, it remains unclear whether warranty and liability disclaimers for open source software will be enforced. UCITA would not only clarify this issue,

---

284. Gomulkiewicz, *supra* note 178, at 190.

285. *See, e.g.*, Maher, *supra* note 183, at 678.

286. Gomulkiewicz, *supra* note 178, at 191.

287. *Id*. at 192 ("If free software authors lose the right to disclaim all warranties and find themselves getting sued over the performance of the programs that they've written, they'll stop contributing free software to the world. It's to our advantage as users to help the author protect this right." (quoting Bruce Perens, author of the Open Source Definition) (internal quotation marks omitted)).

but would give free software developers a leg up on proprietary developers by exempting them from certain implied warranties.

Of particular concern for commercial software developers is the impact of the copyleft provision of the GPL and other reciprocal licenses. Several authors have expressed concern that copyleft is a trap for the unwary commercial developer because it could contaminate an entire software project and force it to be distributed under the GPL.[288] No doubt, developers that have dipped their toes into the open source waters should make themselves keenly aware of this issue. Yet, there is no judicial guidance on whether this term is enforceable and, if so, to what extent it can bind downstream users. In its current form, UCITA does not specifically address this issue. Nevertheless, its deference to, and enforcement of, license terms would validate the copyleft provision as long as certain notice requirements are satisfied. The copyleft provision is more likely to be enforced under UCITA. However, UCITA would also require open source developers and distributors to conform their distribution practices to its requirements. This would ensure that licensees have notice of copyleft license terms before formation of an enforceable contract. In this sense, UCITA would promote uniformity and certainty with regard to the distribution method of open source software.

Enactment of UCITA would not directly solve the problem of license proliferation. Nevertheless, it would provide clear rules for open source developers and help them make informed decisions when choosing a particular license. It would also assist open source standards organizations such as the OSI by providing clear rules regarding when and under what conditions open source licenses would be enforceable.

One could argue that the open source approach in general, and copyleft in particular, suggests that our current intellectual property regime fails to strike an appropriate balance between propertizing intellectual creations and sustaining a robust public domain of freely available information. There is, of course, a paradox embedded somewhere here. Whatever balance is achieved by pure intellectual property law is tilted in favor of creators and against public access by the use of licenses. Open source software emerges from a culture of free and open access to information, yet exists in and is sustained by the copyright and license regimes. As one author points out, "[t]o promote the open source model, it appears, is to accept the legitimacy of licensing models that the open source model is designed to oppose."[289] Perhaps copyright law should provide an alternative mechanism for authors that wish to make their works freely available, instead of resorting to the copyright-and-then-license approach that the current open source movement utilizes. At present, however, the open source movement does rely on the copyright-then-license regime.

VIII. CONCLUSION

It should now be apparent that the open source movement has provided society with innovative and valued software projects including Linux, Apache, and many others. Open source software has contributed to the progress of the new information

---

288. *See, e.g.*, Thomas M. Pitegoff, *Open Source, Open World: New Possibilities for Computer Software in Business*, BUS. L. TODAY, Sep./Oct. 2001, at 52, 55.

289. Madison, *supra* note 12, at 275-76.

society and has provided a foundation for economic growth. This has occurred despite the substantial legal uncertainties looming over the open source licensing regime. To the extent that the open source movement has the potential to provide society with free public goods, it should be supported by law and public policy.[290]

A uniform law, such as UCITA, could potentially benefit the open source movement by generally validating the licensing regime it relies upon and enforcing warranty and liability disclaimers. The UCITA experience represents one effort to address a species of transaction that is becoming omnipresent in our new information economy—the licensing of computer information. Despite the controversy surrounding its drafting and the opposition to its adoption, UCITA is, at the very least, a resource for thinking about the kinds of issues that should be confronted when dealing with transactions in information.[291] Although reports of UCITA's demise have been ongoing and frequent, these reports might be overstated. Adoption of the UCC was at least as controversial as UCITA has been, but the need for uniformity and clarity in commercial law ultimately led to its acceptance and adoption. Given the ubiquity of the software license transaction in our modern economy, there is pressing need for uniformity and clarity in this area as well.

To the extent that positive law—in the form of a uniform code—should be informed by accepted customs, trade usages, and commercial practices, the open source movement represents an important normative consideration. Regardless of whether the open source approach is a rebirth of original programming tradition[292] or the emergence of a new software development paradigm, it deserves our attention. Some have suggested that computer information transactions have not matured to the point where trade usages, customs, and practices are sufficiently well established to provide a foundation for codification. It has been argued that, at present, it is too early to impose a positive uniform code because of the nascent nature of software and information transaction practices. One author explains that "[i]n many industries which would be governed by [UCITA], it is premature to refer to a usage of trade. Customs are rapidly evolving, and deference to a particular norm at a particular time may not be appropriate."[293] The open source movement might support this position in that it represents a paradigm shift in the software development industry. This kind of *laissez-faire* approach, however, fails to recognize and adequately respond to the ever-increasing advance of technology and the impact of that advance on our legal system. The momentum of technological innovation will necessarily force more frequent and imaginative legal evolution.[294] Additionally, although there is some dispute about the degree of overlap between commercial practices in the sales of goods context and commercial practices in the licensing of information context, existing commercial law

---

290. At least one author agrees. "Linux's development and success prove that 'open-source' programming can produce reliable, technologically sound software with remarkable speed. Viewed in this context, it is vitally important that the law support, even encourage, the development of 'open-source' software." Bobko, *supra* note 196, at 84.

291. Boss, *supra* note 135, at 127.

292. Potter, *supra* note 6, ¶ 7 ("In a way, open source is the re-emergence of the original software distribution model.").

293. Maureen A. O'Rourke, *Progressing Towards a Uniform Commercial Code for Electronic Commerce or Racing Towards Nonuniformity?,* 14 BERKELEY TECH. L.J. 635, 651 (1999).

294. That is "evolution," not "revolution."

has already laid some of the necessary foundation for a uniform code to govern software licensing. Finally, the open source movement shows us that even a constituency that was strongly opposed to adoption of UCITA embraces the licensing regime that UCITA also embraces. It is true that the open source approach shifts the paradigm with respect to sharing of source code. However, much like proprietary software developers, open source developers utilize standard form clickwrap and shrinkwrap licenses to control downstream use of their software. In light of the emergence of the open source movement and its reliance on the software licensing regime, it is now time for the ALI and the states to reconsider UCITA.

Matthew D. Stein