# IP's Problem Child: Shifting the Paradigms for Software Protection

Jacqueline D. Lipton
*University of PIttsburgh School of Law*, jdl103@pitt.edu

## *Articles*

# IP's Problem Child: Shifting the Paradigms for Software Protection

JACQUELINE D. LIPTON*

## INTRODUCTION

Imagine that those pre-law computer science classes are finally paying off, and you have become a successful software developer with a concept for a new software application that will revolutionize life as we know it. You previously worked for a large corporation, MensaSoft, Inc., and now want to make it on your own. You finally complete a prototype of the new product. Now you are on the verge of deciding whether to release it into the open source movement for the good of mankind, or to produce it commercially and pay off some of those old college debts.

Before you do anything with the software, you receive a letter from MensaSoft alleging that your new product infringes various proprietary rights in MensaSoft's software, including patents and copyrights, as well as breach of contract and infringement of trade secrets. What does all this mean for you?

It is not a good scenario for you, but at least you are not alone. Those involved in the computer and software law field face similar concerns and confusions on a daily basis. Although the framework for the legal protection of software appeared settled at the end of the twentieth century, exponential developments in programming methodology and digital information law are causing cracks to appear in its foundations. As a result, it is necessary now to re-evaluate the fundamental bases of software protection law. This is imperative to avoid the chilling of innovations in software development and the entrenchment of inefficient digital information practices which are currently supported by inappropriate applications of intellectual

* Professor and Co-Director, Center for Law, Technology & the Arts, Case Western Reserve University School of Law. Thanks to Professors Peter Menell and Pamela Samuelson for commenting on an earlier draft of this Article. All mistakes and omissions are my own.

property law.

This Article advocates such a re-evaluation, with a focus on eliminating, or at least minimizing, inappropriate uses of copyright law in the software context. It suggests that, to some extent, the current emphasis on software patenting issues masks some more serious underlying problems involving software copyrights. Software copyrights are easy to obtain and assert against competitors even though incremental development, including some measure of copying, is fundamental to advances in computer processing. Hence, copyrights can serve to chill innovation unless clearer guidelines about copyright limitations in the software context are developed. The focus of this Article is on identifying some such limitations with specific reference to the scaling back or elimination of copyright protection for software.

Part I identifies how copyright law is intended to operate in the software code context. Part II examines the nature of computer software and recent developments in computer programming methodology that have an impact on questions of copyright protection for such code. Part III analyzes recent legal developments, notably the enactment of the anti-circumvention provisions of the Digital Millennium Copyright Act (DMCA), that have had troubling practical consequences in the software copyright context. Part IV suggests reform options, focusing on the elimination of copyright protection for software code and reliance instead on other means of protection such as trade secrecy, augmented by sophisticated modern digital rights management (DRM) measures. Part V sets out some conclusions on the issue of software copyrightability and identifies areas where more work needs to be done to create an appropriate level of legal protection for software more generally.

Before considering the application of copyright law to software, it is worth briefly making some introductory comments about why this Article has taken copyright as its focus rather than patent law. Many would argue patents cause more potential problems in this context than copyrights in terms of impeding innovation.[1] Why has so much attention been focused on the perils of software patenting and not software copyrighting? Perhaps this is because of the powerful scope of patent rights. Patents grant patentees exclusive rights to exploit inventions. These rights can be enforced against the whole world,[2] whereas software copyrights are limited to protecting the rights-holder against substantial *copying* of relevant code.[3] There are also more defenses available to a

1. *See, e.g.*, Éloïse Gratton, *Should Patent Protection Be Considered for Computer Software-Related Innovations?*, 7 COMP. L. REV. & TECH. J. 223, 235–40 (2003).

2. *See* 35 U.S.C. § 271(a) (2006) (providing that infringement of patent is committed by anyone who "makes, uses, offers to sell, or sells any patented invention" without authorization).

3. 17 U.S.C. § 106 (2006) (defining exclusive rights in copyright works in terms of rights to reproduce and perform protected works). The underlying difference in the policy basis between

person or entity accused of copyright infringement than patent infringement. The fair use defense[4] in particular mitigates against overbroad use of software copyrights to stifle competition in relevant markets.[5] Further, there are limitations to the reach of copyright law under the doctrines of merger and *scènes à faire*[6] that might appear to make copyright law less problematic than patents in impeding innovations in software development.[7]

However, the view that copyright law is less problematic than patent law assumes that most lawyers, judges, and industry players understand the scope of copyright protection with all these inherent limitations. This is likely not the case. Copyrights are obtained much more easily than patents and require very little effort on the part of the copyright applicant. Copyright protection arises when an author affixes her work in a tangible medium of expression.[8] Registration is not required to assert a copyright.[9] A copyright holder, having gone to little effort to secure the copyright, may use her asserted rights to create a chilling effect in a particular market by threatening any competitors that may potentially be

---

copyright and patent is that copyright protection is limited to the fixed literal expression of code. *Id.* § 102. Patents, on the other hand, are granted for inventions generally and these inventions may incorporate, or even be comprised of, software code. *See* Diamond v. Diehr, 450 U.S. 175, 177, 192–93 (1981) (upholding validity of patent involving the use of a programmed digital computer); State St. Bank & Trust Co. v. Signature Fin. Group Inc., 149 F.3d 1368, 1375 (Fed. Cir. 1998) (upholding validity of a computer program for a method of doing business); *In re* Alappat, 33 F.3d 1526, 1536–37 (Fed. Cir. 1994) (upholding validity of waveform display in a digital oscilloscope).

4. 17 U.S.C. § 107 (2006); MARSHALL LEAFFER, UNDERSTANDING COPYRIGHT LAW § 10.01 (4th ed. 2005) ("The doctrine of fair use is a judicially created defense to copyright infringement that allows a third party to use a copyrighted work in a reasonable manner without the copyright owner's consent. Although codified in the 1976 Act, the doctrine of fair use has retained its nature as an equitable rule of reason to be applied where a finding of infringement would either be unfair or undermine 'the progress of science and the useful arts.'").

5. In fact, specific applications of the fair use defense in the computer software context have developed over the years, notably to protect those who need to copy software in order to create an interoperable program. *See* Sega Enter. Ltd. v. Accolade, Inc., 977 F.2d 1510, 1514 (9th Cir. 1993) (holding that decompilation of a computer program, involving copying of the program, in order to produce a compatible, non-infringing program is a fair use); LEAFFER, *supra* note 4, § 10.13 (discussing fair use in the software reverse engineering context).

6. LEAFFER, *supra* note 4, § 2.14[B][4] ("The merger doctrine has been applied primarily to works of utility, such as forms, rules, and computer programs. Under the merger doctrine, courts will not protect a copyrighted work from infringement if the idea underlying the work can be expressed only in one or few different ways, for fear that there may be a monopoly on the underlying idea. In such an instance, it is said that the work's idea and expression 'merge.' Under the related doctrine of *scènes à faire*, courts will not protect a copyrighted work from infringement if the expression embodied in the work necessarily flows from a commonplace idea.").

7. *See* discussion of merger and *scènes à faire* doctrines, *infra* Part I.B.2–3.

8. 17 U.S.C. § 102(a); LEAFFER, *supra* note 4, § 7.01.

9. However, registration is necessary in the United States in order to bring an infringement action. *See* LEAFFER, *supra* note 4, § 7.01. In the computer software context, the author is usually the programmer, although that person can clearly transfer copyright ownership to another person or entity, such as her employer.

infringing upon her copyright. Although a fair use defense or a merger argument may be available to the competing entity, that person or business may not be sufficiently versed in the scope and nature of software copyrights to utilize these arguments and rebut claims made in a demand letter. That person may not even realize that, in order to infringe a copyright, she would have needed an opportunity to peruse and copy the relevant code.[10] Additionally, even if the competitor has the wherewithal to withstand the demand letter and face the prospect of litigation, many judges may not be sufficiently familiar with the operation of software copyright law to reach the "right" result.

The software copyright picture becomes even more complicated when this initial uncertainty is coupled with new programming methodologies, such as object-oriented programming,[11] that rely on copying and reusing code as part of the underlying programming paradigm. While copyright law strives to regulate unauthorized copying, modern programming methods advocate copying and reuse as a fundamental pillar of software design. This makes copyright an even less attractive proposition for software protection if the aim is to encourage innovation.

While some companies in the computer software industry are unquestionably flourishing in today's marketplace, they may be doing so by taking advantage of competitors who lack the wherewithal to combat software copyrights. These large software companies also utilize questionable software patents, restrictive DRM,[12] and contractual measures to stifle competition. All of these barriers may be standing in the way of the incremental developments essential for software innovation.[13] Given the advances in these other methods of protecting software in recent years, software copyrights are less necessary today

10. This is because copyright does not protect an author against independent reinvention of the same or a similar work, but only against actual copying. *See id.* § 2.07[B].

> [I]n copyright law all that is required for protection is independent creation, not striking uniqueness, ingenuity, or novelty. Thus, nothing prevents a valid claim of copyright on two or more substantially similar works so long as they were independently created. An action for copyright reflects this principle by requiring the copyright owner to prove both substantial similarity and copying. Unlike a case for patent infringement, proving substantial similarity alone will not be enough to prove copyright infringement; one must prove by direct or circumstantial evidence that the infringer actually copied another's work.

*Id.* (citations omitted).

11. For a detailed description of the theory underlying object oriented programming methodologies, see *infra* Part II.B.3.

12. *See generally* Declan McCullagh & Milana Homsi, *Leave DRM Alone: A Survey of Legislative Proposals Relating to Digital Rights Management Technology and Their Problems*, 2005 MICH. ST. L. REV. 317, 318 ("Digital rights management is a general term that refers to technology-based protections that permit a rights holder to restrict a user's access to and control of digital content.").

13. Pamela Samuelson et al., *A Manifesto Concerning the Legal Protection of Computer Programs*, 94 COLUM. L. REV. 2308, 2330–32 (1994) (describing ways in which innovation in computer programming is largely incremental and cumulative in character).

than in the past. Lawmakers in the future should consider scaling back copyright protection for software to the point where it is eliminated in all but the most obvious cases of large-scale verbatim copying of code. Even then, there may be some question as to the copyrightability of relevant code because of uncertainties in the application of the merger and *scènes à faire* doctrines. Thus, retaining any level of copyright protection for code will require clearer guidelines in the future about the level of originality required.

The elimination, or at least scaling back, of copyright protection for software would allow more attention to be paid to more appropriate avenues of legal and technological protection for code, including appropriately tailored software patenting policies, coupled with sophisticated DRM measures and contractual licenses. Unlike copyright, all of these avenues focus on protecting the functional value of software and all of them require software developers, who seek to assert proprietary interests in their code, to go to some personal cost and effort to put effective protection measures in place. In this context, it is important to appreciate that contractual and DRM measures are much more sophisticated and reliable from the software developer's point of view today than they were when copyright was first employed to protect code.

Whatever approach is ultimately taken to the question of software copyrightability, Congress, the judiciary, and, to some extent, the United States Patent and Trademark Office (USPTO), likely will have to address potential problems relating to the over-protection of software through means other than copyright. With so many avenues of protection currently open to software developers and so little consensus as to the exact scope of each measure, it is difficult to resolve problems of over-protection without first unraveling other options and examining them individually. Perhaps unraveling problems relating to software copyrights and removing the specter of inappropriate copyright use in the software context can be regarded as an important first step within this larger inquiry. Copyright law is a good starting point because copyright is arguably the least-suited paradigm for protecting software and because it has been relied on so heavily as a means to provide inexpensive and immediate protection to software developers.[14]

## I. COPYRIGHTING CODE

### A. COPYRIGHT FUNDAMENTALS

Copyright protection for software is intended to be quite limited in

14. *See* Peter Menell, *An Analysis of the Scope of Copyright Protection for Application Programs*, 41 STAN. L. REV. 1045, 1072 (1989).

operation and, in particular, to protect only fixed literal elements of software code. To understand why this causes confusion in modern software copyrighting practice, it is necessary to understand why copyright law was originally used as the principal means for the legal protection of software, and some of the intended limitations of copyright law and practice. One of the more obvious limitations relates to the relatively slow development of legal precedent in this area in the face of rapid innovations in the software industry. While copyright law facially might have seemed an appropriate avenue of protection of code in the 1980s and 1990s,[15] more recent advances in programming methodology have moved computer code further and further away from the kinds of literary work that copyright law has historically protected.

Copyright law has traditionally protected original works[16] fixed in a physical medium.[17] The boundaries of the originality requirement have been difficult to ascertain over the years. The consensus is that the bar is not particularly high,[18] at least as compared with the novelty[19] and non-obviousness[20] requirements in patent law. The boundaries of the fixation requirement also posed some difficulties, particularly in the early days of the personal computer revolution, but it is now well settled that software code stored on a digital storage medium[21] will be sufficiently fixed to attract copyright protection.[22]

The inherent characteristics of copyright law pose problems for its

15. *See* Jane Ginsburg, *Four Reasons and a Paradox: The Manifest Superiority of Copyright Over Sui Generis Protection of Computer Software*, 94 COLUM. L. REV. 2559, 2559 (1994); Menell, *supra* note 14, at 1047 (noting the National Commission on New Technological Uses of Copyright Works (CONTU) made a rough empirical judgment in 1978 that copyright would best promote invention and development in the software industry); John Swinson, *Copyright or Patent or Both: An Algorithmic Approach to Computer Software Protection*, 5 HARV. J.L. & TECH. 145, 146 (1991) (proposing a new copyright law test to protect software code).

16. 17 U.S.C. § 102(a) (2006) (noting copyright subsists "in original works of authorship"); LEAFFER, *supra* note 4, § 2.07[B] (stating that for copyright purposes, an original work is one that is independently created and owes its origin to the author, rather than being copied from another).

17. 17 U.S.C. § 102(a) (noting copyright subsists "in original works of authorship fixed in any tangible medium of expression"); LEAFFER, *supra* note 4, § 2.02 (describing the fixation requirement).

18. *See* Feist Publ'ns, Inc. v. Rural Tel. Serv. Co., 499 U.S. 340, 358–59 (1991) (concluding that in the copyright context, a vast majority of compilations will pass the copyright originality test, but "a narrow category of works in which the creative spark is utterly lacking or so trivial as to be virtually non-existent" cannot sustain a valid copyright on this basis); *see also* WILLIAM CORNISH & DAVID LLEWELYN, INTELLECTUAL PROPERTY: PATENTS, COPYRIGHTS, TRADE MARKS AND ALLIED RIGHTS 389 (5th ed. 2003) (noting in the United Kingdom, the originality standard has been described as "de minimis").

19. 35 U.S.C. § 102 (2006). 35 U.S.C. § 101 also requires an invention to be "new and useful" for the subject matter patentability test.

20. *Id.* § 103.

21. Such as a computer's hard drive, a floppy disc, CD, or DVD.

22. MAI Sys. Corp. v. Peak Computer, Inc., 991 F.2d 511, 519 (9th Cir. 1993) (finding that loading copyrighted software into a RAM memory in a computer would constitute fixation for copyright purposes); LEAFFER, *supra* note 4, § 2.03[B].

application to software protection. Copyright protection lasts for a fixed statutory period, currently the author's life plus seventy years in most jurisdictions, including the United States.[23] This may not be the best fit for an industry that thrives on fast paced incremental innovations.[24] Copyright protection is also very cheap and easy to obtain. In most jurisdictions, no formalities are required for a valid copyright to come into existence, although registration is necessary in the United States for enforcement purposes.[25]

One of the limitations of copyright law, particularly in the software code context, is that it only protects against activities related to unauthorized reproductions of a protected work.[26] It does not prevent independent reinvention of a work or prohibit the creation of a product that functions similarly to the original, but uses a different literal expression. The latter situation is not uncommon in the software industry[27] and can arguably lead to under-protection of software code. Copyright will only protect code in cases of exact duplication of code.[28] While these situations unquestionably have the potential to be market-destructive,[29] they are a very narrow sub-class of all of the concerns raised by software developers.

## B. COPYRIGHT LIMITATIONS

### *1. Fair Use and Reverse Engineering*

A brief consideration of some of the relevant aspects of copyright law evidences that the initial intentions of Congress and the judiciary were only to provide limited protection to software through copyright law. This part outlines the ways in which copyright law has been theoretically limited in application to software, while the following parts show how these theoretical limitations are beginning to break down in practice. The later parts evidence ways in which software is now arguably receiving greater protection than originally intended by copyright law.

23. *See, e.g.*, 17 U.S.C. § 302(a) (2006).

24. *See* Samuelson et al., *supra* note 13, at 2357 (suggesting the then-seventy-five year term of protection was too long for the needs of the software industry).

25. *See* Judith Szepesi, *Maximizing Protection for Computer Software*, 12 SANTA CLARA COMPUTER & HIGH TECH. L.J., 173, 188 (1996) ("Neither publication nor registration is necessary in order to receive [copyright] protection. No investment beyond the creation and fixation of the work is required for copyright protection. However, registration is a prerequisite for filing suit in federal court. Special damages can also be collected if the copyright is registered.").

26. *See* 17 U.S.C. § 106 (2006); LEAFFER, *supra* note 4, § 1.05[A] (enumerating exclusive rights under copyright and their limitations).

27. *See* Samuelson et al., *supra* note 13, at 2317 ("A second comer can develop a program having identical behavior, but completely different text through a process sometimes referred to as 'black box' testing.").

28. *Id.* at 2421.

29. *Id.* (noting exact duplications of code are the most market-destructive appropriations of program behavior).

The main theoretical limitations imposed on software protection through copyright law arise under the fair use defense, the merger doctrine, the *scènes à faire* doctrine, and the copyright misuse doctrine. These aspects of copyright law are the subject of this part.

In a nutshell, the fair use defense has its origins in the common law,[30] although it has now been loosely codified in Title 17 of the United States Code.[31] The defense allows a third party to use a copyrighted work in a reasonable manner without the express permission of the copyright owner.[32] The main significance of the doctrine in the software copyright context arises in debates regarding the extent to which a competitor of a software developer should be entitled to reverse engineer a computer program.[33] Courts have held that disassembly or decompilation of a computer program to produce a compatible, non-infringing program can amount to fair use in the copyright context.[34]

Studying reverse engineering does shed some light on the level of protection software developers can realistically expect from copyright law. As noted by Professor Samuelson, it is very easy for a competitor to develop a program with identical functional behavior to the original but with completely different underlying literal code.[35] Specific forms of reverse engineering, such as "black box" testing, can be used to achieve such results.[36] This kind of reverse engineering avoids even the need to invoke the fair use doctrine because it relies on different teams of software engineers at different stages of the process to avoid anyone from ever making a literal copy of the original program during the process.[37] Thus, reverse engineering is excused in various ways under copyright law. Some literal copying is excused under the fair use doctrine to make compatible, non-infringing programs,[38] while reverse engineering not involving literal copying does not invoke copyright law at all.[39]

30. LEAFFER, *supra* note 4, § 10.01 ("The doctrine of fair use is a judicially created defense to copyright infringement....").

31. This section was inserted in the 1976 revisions to the Act. *See* 17 U.S.C. § 107 (2006).

32. *See* LEAFFER, *supra* note 4, § 10.01.

33. *See id.* § 10.13.

34. Sega Enter. Ltd. v Accolade, Inc., 977 F.2d 1510, 1520 (9th Cir. 1993); LEAFFER, *supra* note 4, § 10.13; *see also* MARK LEMLEY ET AL., SOFTWARE AND INTERNET LAW 136–38 (2d ed. 2003).

35. Samuelson et al., *supra* note 13, at 2317.

36. *Id.*

37. *Id.* at 2317–18. ("'[B]lack box' testing... involves having a programmer run the program through a variety of situations, with different inputs, making careful notes about its behavior. A second programmer can use this description to develop a new program that produces the specified behavior (i.e., functionally identical to the first program) without having access to the text of the first program, let alone copying anything from it. A skilled programmer can, in other words, copy the behavior of a program exactly, without appropriating *any* of its text." (citations omitted)).

38. *Sega*, 977 F.2d at 1520; LEAFFER, *supra* note 4, § 10.13; *see also* LEMLEY ET AL., *supra* note 34.

39. *See, e.g.*, Samuelson et al., *supra* note 13, at 2317–18.

### 2. *Merger*

Copyright's merger doctrine is also potentially relevant in the computer software context. However, this doctrine is somewhat underutilized, possibly because its scope of operation is not well understood in general,[40] let alone in the specialized software development context. The merger doctrine traces from the 1880 case of *Baker v. Selden*.[41] As explained by Professor Leaffer:

> [T]he doctrine of *Baker v. Selden* implies that there are some instances where the use of a system or process necessitates the identical copying of the author's expression of the system or process. In other words, if the underlying idea (or system, process, or method of operation) can effectively be expressed in only one way, the idea and expression are said to have "merged." When this occurs, the work cannot receive protection under copyright law. To allow copyright protection in such an instance would undermine the notion that ideas are in short supply and their protection is not worth the social costs of the monopoly. It would also blur the distinction between copyright and patent law.[42]

As software programming methodology has developed over the last decade or so, it is arguable that the merger doctrine should be increasingly applied to many disputes involving alleged software copyright infringements. In fact, some cases are beginning to identify the merger doctrine as very significant in software copyright infringement actions, particularly with respect to simple, functional software applications like digital encryption devices.[43] The following discussion explains in more detail why recent developments in programming methodology necessitate that courts and legislatures take a more detailed look at the potential application of the merger doctrine in the software code context.

Briefly, as computer programs are becoming more modularized[44] and object-oriented,[45] more situations are likely to arise where ideas and expressions are merged. Some judges have recently suggested that in order to attract copyright protection,[46] some kinds of code will need to be

40. LEAFFER, *supra* note 4, § 2.14[B][2] (describing some apparent inconsistencies of application of the merger doctrine).

41. 101 U.S. 99 (1880).

42. LEAFFER, *supra* note 4, § 2.14[B][2].

43. *See, e.g.*, Lexmark Int'l, Inc. v. Static Control Components, Inc., 387 F.3d 522 (2004).

44. The concept of modularization in the computer programming sense is explained *infra* Part II.B.2.

45. The theory of object-oriented programming is explained *infra* Part II.B.3.

46. This discussion assumes that a merger of idea and expression will negate the possibility of copyright protection ab initio for a particular program or section of code. However, it is worth noting that there is arguably a federal circuit split as to whether the merger doctrine functions as a bar to copyrightability or rather as a defense to infringement, explaining a substantial similarity between two works. *See Lexmark*, 387 F.3d at 557 (Feikens, J., concurring in part and dissenting in part) (noting that it is unclear whether the merger doctrine operates as a bar to copyrightability or rather as a defense to particular types of infringement).

drafted in a more unusual or original manner to avoid the merger doctrine.[47] This approach might ultimately lead to inefficiencies in programming because more unusual approaches to the programming of simple functions are not likely to achieve the most efficient functional results. Thus, a robust application of the merger doctrine in some areas of software copyright law might emphasize the inappropriateness of copyright law to protect the valuable aspects of software. Nevertheless, the application of the merger doctrine is certainly appropriate in cases where a software developer is attempting to utilize copyright law to gain a long term monopoly over an extremely simple and functional module of code.

### *3.* SCÈNES À FAIRE

Similar points may be made about the *scènes à faire* doctrine as applied to computer code, although the application of this doctrine has, to date, proven to be even more uncertain than the merger doctrine. The *scènes à faire* doctrine is a judicially created aspect of copyright law that deals with problems arising from the fact that, in certain circumstances, it is virtually impossible to create a work without employing stock standard characters, motifs, scenes, etc.[48] Under the *scènes à faire* doctrine, courts will not uphold copyrights if the relevant expression necessarily derives from a commonplace idea.[49] The most obvious applications are to literary, artistic, and musical works such as novels, plays, movies, and songs, where there are a limited number of ways to express a particular narrative element.[50] However, there is no reason why similar ideas cannot be applied in the software context. There is an increasing jurisprudence suggesting that judges are prepared to consider elements of code as stock ways of expressing a particular programming idea.[51] Thus, copyright protection would not be available to these stock elements.[52]

47. *Id.* at 551–52 (Merritt, J., concurring) (expressing concern that future manufacturers will draft more "creative" software code to avoid holdings such as the appeal decision in *Lexmark* that deny copyright protection for lock out codes).

48. Douglas Lichtman, *Copyright as a Rule of Evidence*, 52 DUKE L.J. 683, 738 (2003) (noting that *scènes à faire* is a judicially created doctrine).

49. *See* LEAFFER, *supra* note 4, § 2.14[B][4].

50. *See* B. MacPaul Stanfield, *Finding the Fact of Familiarity: Assessing Judicial Similarity Tests in Copyright Infringement Actions*, 49 DRAKE L. REV. 489, 508–10 (2001) (discussing cases involving literary, movie, and music copyrights and the application of the *scènes à faire* doctrine).

51. *See Lexmark*, 387 F.3d at 535–36 (surveying case law and commentary on the application of the *scènes à faire* doctrine in the software code context).

52. It is not clear whether the *scènes à faire* doctrine is intended to function purely as a defense to an infringement action, or whether it potentially negates the copyrightability of the claimed subject matter ab initio. A leading copyright treatise suggests that the application of *scènes à faire* does not negate the copyrightability of the relevant elements of a work, but rather explains, presumably by way of defense, any substantial similarity between two works. MELVILLE B. NIMMER & DAVID NIMMER, 4 NIMMER ON COPYRIGHT § 13.03[B][4] (2006); *see also Lexmark*, 387 F.3d at 559 (Feikens, J., concurring in part and dissenting in part) (noting that there is currently a circuit split on this question, as with the

### 4. *Copyright Misuse*

As noted above, this Article advocates eliminating, or at least significantly scaling back, copyright protection for software because it is a bad fit with the needs of the software community. Alternatively, the operation of doctrines such as merger and *scènes à faire* in the software context should be clarified to avoid over-protection of software under copyright law. However, it is worth briefly noting the copyright misuse doctrine as another potential avenue for combating potential over-protection of software through copyright law. The copyright misuse doctrine is a somewhat troubling aspect of copyright law, the scope of which is not well understood in practice.[53] It derives from the patent misuse doctrine that began as a defense to a patent infringement suit and has been used to prevent patent owners from abusing market power to hinder competition through tying[54] and other restrictive licensing arrangements.[55] The idea of exporting a similar concept into copyright law has been based on the same policy arguments as those underlying patent misuse: that is, the idea of seeking to increase the store of human knowledge and expression by encouraging scientific and artistic innovation without conferring monopoly power over property not directly subject to a copyright or patent.[56]

Professor Leaffer has suggested that copyright misuse should be more limited in operation than patent misuse because of differences between the nature of copyrights and patent rights.[57] He has noted that copyrighted works are generally highly substitutable and that many different songs, films, or computer programs may compete at the one time for consumers' dollars.[58] Given this fact, the concern about unfair monopolies in these products is less likely to arise than in the patent context, where patents are more likely to contribute to the development of an economic monopoly in a particular patented product.

merger doctrine); *cf.* Stanfield, *supra* note 50, at 508 (noting that *scènes à faire* doctrine separates copyrightable elements of a work from non-copyrightable settings, sequences of events, and stock themes that arise in a particular work).

53. *See generally* Dan Burk, *Anticircumvention Misuse*, 50 UCLA L. REV. 1095, 1124-31 (2003); Ilan Charnelle, *The Justification and Scope of the Copyright Misuse Doctrine and its Independence of the Antitrust Laws*, 9 UCLA ENT. L. REV. 167, 167 (2002); Brett Frischmann & Dan Moylan, *The Evolving Common Law Doctrine of Copyright Misuse: A Unified Theory and Its Application to Software*, 15 BERKELEY TECH. L.J. 865, 868 (2000); Karen E. Georgenson, *Reverse Engineering of Copyrighted Software: Fair Use or Misuse?*, 5 ALB. L.J. SCI. & TECH. 291, 312-20 (1996).

54. Tying is an antitrust practice involving the supply of one good or service on condition that the purchaser also buys a second good or service that is therefore effectively tied to the first. *See, e.g.*, Robert W. Emerson, *Franchising and Consumers' Beliefs About "Tied" Products: The Death Knell for* Krehl?, 45 FLA. L. REV. 163, 165-80 (1993).

55. LEAFFER, *supra* note 4, § 10.21[D][1].

56. *Id.*

57. *Id.*

58. *Id.*

In the computer program context in particular, with the ability of competitors to reverse engineer programs as noted above, many substitutes are likely. At least this would technically be the case in the absence of other effective legal and practical avenues of protection including patents, DRM measures, and restrictive contractual licenses that have a negative impact on a competitor's ability to reverse engineer. Thus, it may be that copyright, *coupled with* restrictive DRM measures and contractual licenses, leads to a situation that might amount to copyright misuse because it effectively bars competitors from creating competing products thereby giving the original developer a monopoly. Where a developer then uses this monopoly to charge unfair prices or to tie unrelated products and services to the protected software, a strong policy argument for the development of a copyright misuse doctrine is created.[59]

Recent case law does appear to be evidencing moves in this direction, particularly in the wake of the enactment of the DMCA. There has been increased concern that software developers can utilize restrictive DRM measures bolstered by the anti-circumvention and anti-trafficking provisions of the DMCA to shut down competition in markets unrelated to the protected software.[60] This prospect raises a number of complicated issues, most of which are beyond the scope of this Article. For example, would a copyright misuse defense apply to a suit by a copyright owner for infringement of the anti-circumvention and anti-trafficking provisions of the DMCA?[61] Various suggestions have been made in recent years to address these concerns, most of which relate to amending the DMCA to clarify its position on abuse of DRM measures.[62]

This Article takes a different approach.[63] It suggests that if computer

59. Jacqueline Lipton, *The Law of Unintended Consequences: The Digital Millennium Copyright Act and Interoperability*, 62 WASH. & LEE L. REV. 487, 540–42 (2005).

60. *See* Lexmark Int'l, Inc. v. Static Control Components, Inc., 253 F. Supp. 2d 943 (E.D. Ky. 2003), *vacated*, 387 F.3d 522 (6th Cir. 2004); Chamberlain Group, Inc. v. Skylink Techs., Inc., 292 F. Supp. 2d 1023, 1033–40 (N.D. Ill. 2003), *aff'd*, 381 F.3d 1178 (Fed. Cir. 2004); *see also* Burk, *supra* note 53; Lipton, *supra* note 59; ELECTRONIC FRONTIER FOUNDATION, UNINTENDED CONSEQUENCES: FIVE YEARS UNDER THE DMCA *passim* (2003), *available at* http://www.eff.org/IP/DMCA/unintended_consequences.pdf.

61. *See* Burk, *supra* note 53, at 1132–40 (noting that because the DMCA's anti-circumvention right is a different kind of right to traditional copyright in that it prevents "access" as opposed to merely copying, the type of anticompetitive misuse of this right does not implicate patent or copyright misuse); Lipton, *supra* note 59, at 540–42 (stating the potential application of the copyright misuse doctrine in cases involving infringements of the anti-circumvention and anti-trafficking provisions of the DMCA is unclear).

62. *See* Burk, *supra* note 53, at 1132–40 (suggesting the need to develop a new "anticircumvention misuse" doctrine to apply to abuses of the DMCA's anti-circumvention and anti-trafficking provisions by copyright holders).

63. I note that this approach is different to the approach I advocated previously. Lipton, *supra* note 59. In that article, I advocated a presumption against DMCA liability where the plaintiff could not show that the protection of a copyrighted work was a central commercial concern. *Id.* at 521–28. I

code was uncopyrightable per se or, at the very least, the merger and *scènes à faire* doctrines were used to minimize the amount of software copyrighting, questions relating to the scope of the copyright misuse doctrine could largely be avoided altogether, particularly in the DRM context. If relatively stock modules of code could not be copyrighted either because of the elimination of copyright protection for code, or as a result of the application of the *scènes à faire* or merger doctrines, that code could not be used as the basis of a DMCA action that effectively ties an associated product or service to the code. This would avoid the need to rely on a copyright misuse or anti-circumvention misuse doctrine in such contexts. Without copyright protection, a copyright misuse defense would be unnecessary. This Article focuses on removing much stock standard code, if not *all* code, from the ambit of copyright law.

Concerns raised some years ago about the suitability of copyright to protect code clearly still resonate today[64] and, in fact, developments in both copyright law and programming methodology have made the situation much more complex than it has been in the past. In many ways, copyright law is a worse fit for computer software products today than it was in the 1980s and 1990s. In fact, arguably the only really beneficial use of software copyrights, outside of the narrow case of preventing exact duplication of code generally, is in the area of open source software.[65] Open source software relies to a significant extent on the ability to assert copyrights[66] in software and then to attach open source licenses to those copyrights to ensure that no one can monopolize the code without infringing the license.[67] Technically, the same result would likely be possible in the absence of copyright protection for software code. It would just be more cumbersome to draft relevant license provisions if they were not attached to a familiar form of intellectual property right.

am not recanting that position here, but simply suggesting that the DMCA misuse problem is born out of a larger concern about copyrighting software code more generally. The removal of copyright protection for software code as a general matter would render moot the suggestions that some new approach needs to be taken to what Professor Burk would call "anticircumvention misuse." Burk, *supra* note 53, at 1132. However, if copyright protection is to be retained for software code in the longer term, Professor Burk's and my suggestions for reforming the application of the DMCA's anti-circumvention and anti-trafficking provisions would still stand. These are alternative arguments dealing with the problem at different levels of generality.

64. *See* Samuelson et al., *supra* note 13.

65. *Id.* at 2421.

66. Patents are too time consuming and economically cumbersome for the open source movement and, in any event, open source developers generally do not want to rely on patents which tend to be used within commercial markets to generate economic monopolies in particular inventions.

67. Joseph Scott Miller, *Allchin's Folly: Exploding Some Myths About Open Source Software*, 20 CARDOZO ARTS & ENT. L.J. 491, 497 (2002) ("The GPL [General Public License used by the open source movement] demonstrates that one can harness the control that copyright law provides to make a piece of software fully and indefinitely accessible, or free, to its users. The carefully crafted license terms do all the work. In other words, open source software, far from forswearing copyright protection, relies centrally on the basic rights that copyright law gives to authors.").

Even if a stand-alone intellectual property right is needed to support the open source movement, it is questionable whether software copyrights are the most appropriate option here. This might be an area where a new sui generis form of intellectual property right should be tailored to address the needs of the relevant software community.

## II. Emerging Trends in Programming Methodology

Modern advances in software programming add a new dimension to the concerns about software copyrighting raised above. Much of the legal literature on software copyrighting has assumed that the nature of software remains relatively static over time. In fact, the kind of software in existence when copyright protection was first extended to software was significantly different than the kind of software commonly developed today. In many ways, more recent developments in programming methodology make copyright protection even less appropriate for computer code than it has been in the past.

### A. The Anatomy of a Computer Program

#### *1. Design Documentation*

To understand relevant trends in software design methodology, it is important to have a basic understanding of the general structure, or anatomy, of a computer program. The software design process involves a number of different stages during which different program elements are created. When writing a new program from scratch, most programmers will start with some form of design or plan to help them structure the overall program. This could be expressed in a graphical format such as a diagram or flowchart, or it could be in a more verbal form, such as pseudocode.[68] Whatever the precise format, the design will give the programmer an idea of the structure of the program and what sorts of algorithms[69] will be needed.

Obviously, whatever the precise methodology chosen to create the overall program design, it will be in a fixed, literal form of expression. Thus, at first glance, it is likely that these designs, to the extent that they are original, meaning not copied,[70] would generally attract copyright protection in most jurisdictions as either literary or artistic works,

68. Pseudocode is basically a form of "structured English" which looks like the source code of a high-level programming language, but is more generic and perhaps somewhat more easily readable. *See* Lesley Anne Robertson et al., Simple Program Design: A Step-By-Step Approach 6 (3d ed. 2000).

69. *Id.* at 271 ("[Algorithms are a] set of detailed, unambiguous and ordered instructions developed to describe the processes necessary to produce the desired output from given input.").

70. It is not as unusual as one might think to copy flowcharts or pseudocode from other sources to write a new program. Computer programming texts, for example, often contain examples of model structures for common programming problems. *See, e.g.*, Robertson et al., *supra* note 68, at 149–73 (containing general algorithms in pseudocode and flowchart notation for common business problems).

depending on their precise format. Most copyright conflicts have not involved these early design documents.

### 2. *Source Code and Object Code*

More important copyright questions have arisen in relation to the later steps in the software development process, particularly the source code, and, to some extent, the object code. Source code is the set of instructions written by a programmer in a specific computer language. Over the history of computing, a variety of different languages, and different levels of computer languages, have been developed.[71] Because a computer only understands binary numbers (ones and zeros), the earliest forms of computer language were actually written in machine-readable binary code (or "machine language"). In fact, it is still the case today that programs written in more sophisticated languages must be converted into binary code, or machine-readable object code, via an interpreter or compiler in order to be executed by a computer.[72] The next development after machine language was assembly language that replaced ones and zeros with mnemonic codes—abbreviations that were easy for a programmer to remember, such as "MP" for multiply.[73] Assembly languages use a translating device called an assembler to convert source code into machine language so the computer can understand and execute the program.[74]

At the next level of sophistication after assembly language came the high-level languages like FORTRAN and COBOL.[75] These languages represented a significant evolution in computer programming.[76] They were designed for specific kinds of problems and used syntax that would be familiar to people who dealt with those problems most often. FORTRAN was designed to solve mathematical problems while COBOL was designed to solve business problems and used commands and operations that were familiar to business people.[77] Again, these kinds of languages required a translator in the form of a compiler to translate the program to machine-readable object code for the computer to read and execute.[78] Other common examples of these high-level languages include BASIC, C, C++, and Java.[79] These high-level (or third generation)[80] languages are the most commonly used in many modern

71. H.L. CAPRON & J.A. JOHNSON, COMPUTERS: TOOLS FOR AN INFORMATION AGE 447–50 (8th ed. 2004).

72. *Id.* at 448.

73. *Id.*

74. *Id.* at 449.

75. *See id.*

76. *Id.*

77. *Id.*

78. *Id.*

79. *Id.* at 450.

80. In this context, machine language was the first generation language, and assembly language is

commercial applications.

This is where the interesting legal issues began to arise. Once languages adopted source code that was more like a human language and less like binary code, the fixed literal expressions of the code created by programmers looked more like any other literal expressions protected by copyright law. The main difference was their functionality. However creative they were in terms of expression or underlying idea, their commercial value lies ultimately in their functional behavior.[81]

Following the high-level languages came the very high-level languages also known as fourth generation languages (or 4GLs).[82] These languages are distinguishable from the high-level third generation languages in that they are non-procedural languages.[83] Whereas the previous three generations of languages, in fact, were procedural in the sense that they required a programmer to describe a step-by-step procedure to solve a problem, the 4GLs allow a programmer to specify desired results and the language in many cases can develop the solution.[84] Using 4GLs enables much greater productivity because most ideas can be expressed in fewer lines of code than in earlier generation languages.[85]

Computer science is now producing the beginnings of fifth generation languages, which are often described as natural language because of their resemblance to natural spoken or written English.[86] Fourth and fifth generation languages are somewhat beyond the scope of this Article. It is not necessary to look any further than third generation languages[87] to observe that copyright law is an unsuitable paradigm for the legal protection of software code. Nevertheless, the development of these languages is a reminder of the constant advances in information technology that require the law to keep pace with new challenges. It should never be assumed that a field like computer science remains static over time; once a legal paradigm has been adopted, it must be regularly reviewed. Computer science developments occur rapidly and require lawmakers to constantly monitor the legal frameworks employed to protect code.

When considering the nature of source code generally, it becomes obvious why it was possible to apply copyright law as a legal protection

referred to as the next (or second) generation. *See id.* at 448.

81. Samuelson et al., *supra* note 13, at 2319 (noting that programs have almost no value in their fixed literal expression, rather their value lies in their behavior).

82. CAPRON & JOHNSON, *supra* note 71, at 449.

83. *Id.*

84. *Id.*

85. *Id.*

86. *Id.* at 450.

87. In fact, third generation languages are still the most commonly used languages in modern programming despite recent advancements in fourth and fifth generation programming. *See* DAVID MCCOMB, SEMANTICS IN BUSINESS SYSTEMS 140 (2003).

mechanism. Source code does in some ways resemble the traditional subject matter of copyright law if it is sufficiently original in its expression and is affixed in a permanent form. If we accept that a work can be fixed in the memory of a computer,[88] then we seem to have something that at least looks like a literary work. This view has certainly been accepted internationally[89] and has been adopted in most national copyright legislation.[90]

With regard to concerns about the value of software being in its functionality rather than its literal code, Professor Menell in the late 1980s noted that it was surprising, although perhaps understandable, that copyright would be adopted as a key form of protection during the early legal debates on software.[91] He noted that the majority of the National Commission on New Technological Uses of Copyright Works (CONTU) believed that the advantages of copyright for software protection outweighed the disadvantages of other methods of protection such as patent and trade secret law.[92] The advantages of copyright protection included the lack of formalities for protection and the lengthy duration of protection.[93] The disadvantages of patent and trade secret protection included doctrinal and practical difficulties inherent in obtaining patent protection, and the ease with which trade secret protection for software could be lost.[94]

Subsequent history has shown that patent and trade secrecy have not proven as problematic as CONTU feared. Patents have readily been granted over software and software-related inventions in the United States.[95] While initially more problematic, trade secrecy has become a much more viable option due to modern sophisticated DRM measures and enforceable contractual licensing provisions.[96]

88. MAI Sys. Corp. v. Peak Computer, Inc., 991 F.2d 511, 517–19 (9th Cir. 1993) (holding that a recordation or writing needs to last a long time to attract copyright protection and loading copyrighted software into a RAM memory in a computer constitutes fixation for copyright purposes); LEAFFER, *supra* note 4, § 2.03[B].

89. *See* World Intellectual Property Organization Copyright Treaty, art. 4, Dec. 20, 1996, S. TREATY DOC. NO. 105-17 (2002), 36 I.L.M. 65 [hereinafter WCT] (providing that computer programs are to be protected as literary works under copyright law whatever their form of expression).

90. *See, e.g.*, Copyright, Designs and Patents Act, 1988, c. 48, § 3(1)(b) (Eng.) (defining a computer program as a literary work for copyright purposes); Copyright Act, 1968, § 10(1) (Austl.) (including computer programs in the definition of "literary work").

91. Menell, *supra* note 14, at 1072.

92. *Id.*

93. *See id.*

94. *Id.*

95. World Intellectual Property Organization, Business Method and Computer Software Patents, http://www.wipo.int/sme/en/e_commerce/computer_software.htm (last visited Nov. 5, 2006) ("Today, there are an increasing number of software and business methods which are protected by patents in the United States . . . .").

96. *See* ProCD, Inc. v. Zeidenberg, 86 F.3d 1447, 1449 (7th Cir. 1996) (upholding clickwrap and shrinkwrap license applied to computerized database); *In re* RealNetworks, Inc., Privacy Litig., No. 00-

Although not technically within the ambit of many of the early debates on legal protection of software, it is worth noting that there appears to be no necessary bar to the protection of object code through copyright law.[97] It is an original literal expression, albeit consisting of only ones and zeros. There appear to be few situations where a software developer today would seek to assert copyright in object code. Most of the concerns have been with human-readable source code that can be reverse-engineered or copied. Nevertheless, as object code is merely the machine-readable form a computer compiled using source code, it could presumably attract most of the same forms of legal protection as source code.

### *3. Program Documentation and Program Output*

Two other aspects of a computer program deserve brief mention before considering in more detail the question of how emerging trends in programming methodology affect the copyright question. The first of these is program documentation, including operating manuals and programmers' notes and annotations to relevant code.[98] The second is the form of output produced by a program, notably any graphical user interface (GUI) generated by an operating system or applications program.[99] A GUI is basically what the user sees on the computer screen as she executes a program.[100] Both of these aspects of programming raise legal protection concerns, but they are largely outside the scope of this Article, which focuses on the unsuitability of copyright protection for the basic building blocks of a program, its source code. They are mentioned here for completeness and to avoid confusion.

The question about program documentation is relatively simple. Fixed written documentation will be copyrightable provided it is sufficiently original for copyright purposes.[101] This may be different for documentation existing within a program itself. In many computer languages there is a mechanism for annotating code as a programmer is

C-1366, 2000 WL 631341, at *5 (N.D. Ill. May 8, 2000) (holding that web users had agreed to an arbitration agreement in a clickwrap license); Julie Cohen, *Copyright and the Jurisprudence of Self-Help*, 13 BERKELEY TECH. L.J. 1089, 1090 (1998) (commenting on the use of self-enforcing digital contracts to protect their "informational rights" as opposed to reliance on copyright law); Tarra Zynda, Ticketmaster Corp. v. Tickets.com, Inc.*: Preserving Minimum Requirements of Contract on the Internet*, 19 BERKELEY TECH. L.J. 495, 505 (2004) (noting that courts have generally upheld the validity of shrinkwrap and clickwrap licenses applied to computer software).

97. CORNISH & LLEWELYN, *supra* note 18, at 765.

98. ROBERTSON ET AL., *supra* note 68, at 104; Menell, *supra* note 14, at 1051.

99. CAPRON & JOHNSON, *supra* note 71, at 67–68; Menell, *supra* note 14, at 1071, 1089. An operating system is computer software that controls the operation of various programs running on a computer. It allots processing time for each program in turn and can handle multiple users on the same system. An applications program is a specific program executed by a computer user. Familiar examples include word processing programs, database programs, spreadsheet programs, etc.

100. CAPRON & JOHNSON, *supra* note 71, at 67–68.

101. Menell, *supra* note 14, at 1047–48.

writing it. This helps the programmer to remember her thinking behind the various lines of code.[102] Documentation that simply *describes* how a program works is unlikely to be patentable as it is unlikely to rise to the level of innovation required for patentability, despite the fact that it might describe or explain something else that is patentable.

Some more difficult legal issues have arisen with respect to GUIs.[103] An obvious example of a GUI with which most of us are familiar is the design and format of the Windows desktop distributed by Microsoft, including the design of specific icons, which are small graphic designs a user clicks with a mouse to access a particular program. As Professors Cornish and Llewelyn noted, GUIs are "closely allied to programs, but (at least for copyright purposes) must be considered something apart."[104] Professor Menell noted early in the days of the software protection debates that GUIs may well be protected separately from the source code that generates them.[105] They might be protected as literary works, pictorial or graphic works, or audiovisual works.[106] The form of protection will depend on their content.[107]

Screen displays of the kinds described above would certainly appear to meet the requirements of copyrightability provided we accept, as we did for source code, that they are in a sufficiently permanent form even though they only appear on screen while a program is being executed. Like source code, however, and unlike more traditional copyright works, many GUIs and their constituent elements serve a significant *functional* purpose. Computer users become familiar with methods for interacting with GUIs and there is a value in ease of use and familiarity to consumers. The ability to click on an icon with a mouse to run a program has a certain utility and commercial value, in contrast to older form command line interfaces where users needed to know specific commands to run programs. A user-friendly GUI certainly makes programs more appealing to consumers and therefore more commercially valuable. There is also significant pressure on competitors of an original software manufacturer to create programs that are similar, in the sense that they

102. *See* ROBERTSON ET AL., *supra* note 68, at 3.

103. There is an associated question relating more generically to computer generated outputs from executing a program. The question has also arisen whether a work can attract copyright protection if it is computer generated in the sense that there is no actual human author of the work. In the United Kingdom, the Copyright, Designs and Patents Act, 1988, c. 48, § 9(3) (Eng.), provides that in the case of computer-generated literary, dramatic, musical, or artistic work, the author is the person who undertook the arrangements necessary for the creation of the work. *See also* Copyright, Designs and Patents Act, 1988, c. 48, § 178 (Eng.) (defining "computer-generated"); CORNISH & LLEWELYN, *supra* note 18, at 399.

104. CORNISH & LLEWELYN, *supra* note 18, at 776.

105. *See* Menell, *supra* note 14, at 1089.

106. *Id.*

107. *Id.*

function in a similar fashion to existing programs on the market with which most consumers are familiar. Thus, if most consumers are familiar with the Windows operating system and its use of graphical features such as framed windows, drop down menus, point and click commands, and functions represented by icons, Windows competitors will want to emulate that functionality in their own screen displays.[108]

The ultimate questions about copyrightability of screen displays have not been resolved definitively in any jurisdiction. The judicial authority, such as it is, has been relatively case-specific, so it is difficult to extract a general principle on copyrightability of screen displays. It appears that American courts will only grant "thin" copyright protection to screen displays essentially amounting to a prohibition against verbatim copying.[109] Professor Leaffer has noted that American courts generally will only allow protection for the artistic features of a GUI and deny protection to more clearly functional features.[110] They have also denied protection to features that have become industry standards even if they were arbitrary and fanciful at the time they were originally adopted.[111] It is possible that other jurisdictions may ultimately grant broader intellectual property protection to GUIs. Professors Cornish and Llewelyn have suggested that English courts may be more flexible in protecting GUIs than American courts, although to date, the issue has been considered only in passing.[112] They also note that screen icons in particular may be protectible under registered designs law in jurisdictions that have developed such statutes.[113]

Thus, it is important to appreciate that the law may have to treat different aspects of computer programs differently. There is, for example, an emerging legislation and jurisprudence relating to the protection of traditional copyright works stored digitally, such as digital

108. In fact, Microsoft's Windows system effectively "copied" this functionality from Apple, the corporation that originally developed the windows-based graphical user interface with point and click commands and drop down menus. Apple Computer, Inc. v. Microsoft Corp., 35 F.3d 1435, 1438, 1446 (9th Cir. 1994). Apple sued Microsoft unsuccessfully for copyright infringement in terms of this alleged copying. *Id.* at 1438–39. Apple's suit was unsuccessful because the court held that Apple was really attempting to claim copyright in functional aspects of its system, which were unprotectible. *Id.* at 1439; *see also* CORNISH & LLEWELYN, *supra* note 18, at 777.

109. CORNISH & LLEWELYN, *supra* note 18, at 777 (observing that American courts are not sympathetic to claims to protect GUIs under copyright law); LEAFFER, *supra* note 4, § 2.14[B][3] ("GUIs have received thin copyright protection that amounts to a prohibition against verbatim copying.").

110. LEAFFER, *supra* note 4, § 2.14[B][3].

111. *Id.*

112. CORNISH & LLEWELYN, *supra* note 18, at 776.

113. *Id.* at 776–77. In these jurisdictions, registered designs are a form of statutory generic intellectual property right that is related to, but distinct from, copyright law. *See, e.g.*, Designs Act, 2003 (Austl.); Registered Designs Act, 1949, 13 & 14 Geo. 6. (Eng.). These statues are much broader in scope than American designs law, which is limited to the protection of vessel hull designs. 17 U.S.C. § 1301(b) (2006) (defining protected designs).

music and movies, that must be treated differently from the legal protection of computer source code.[114] Like the GUIs described above, digital music, movies, eBooks, and the like, raise very different kinds of copyright concerns than any underlying source code through which they are produced on a computer system. This Article is concerned with the legal protection of software *code* through copyright law. It does not suggest that *outputs of code*, such as digital audio and video displays, should not be protected against copying through aggressive application of copyright law and associated protections such as those now found in the anti-circumvention and anti-trafficking provisions of the DMCA.[115] This is particularly the case where those outputs are simply digital versions of the kinds of literary and artistic works that were historically protected by copyright law in more traditional formats.[116]

Computer programs break down into a number of distinct elements that may be separately protected by various aspects of the legal system. This Article focuses predominantly on source code, and whether it is finally time to partially or fully scale back the availability of copyright protection for such code. However, that is not to say that copyright will not be a viable form of protection for other elements of computer software, notably various outputs and documentation. It is necessary to understand this distinction. Not all digital age copyright questions are about software *code*. However, some of the more worrying recent digital copyright questions discussed later in this Article evidence the way issues about protection of code have become aggregated with questions about the protection of digital copyright works more generally against unauthorized access and use.

Eliminating software copyrights, or at least dramatically limiting their scope, would alleviate some of these concerns and would allow copyright law to do its job with respect to the kinds of works it is best suited to protecting—those where the value is in the fixed, literal expression. Prior to examining these questions in more detail, it is necessary to consider emerging trends in programming methodology which in and of themselves suggest that copyright is a less suitable protection method for software code than it might have been in previous years.

114. *See, e.g.*, 321 Studios v. Metro Goldwyn Mayer Studios, Inc., 307 F. Supp. 2d 1085, 1092–99 (N.D. Cal. 2004); United States v. Elcom Ltd., 203 F. Supp. 2d 1111 *passim* (N.D. Cal. 2002); Universal City Studios, Inc. v. Reimerdes, 111 F. Supp. 2d 294, 322, 323–24 (S.D.N.Y. 2000), *aff'd sub nom.* Universal City Studios, Inc. v. Corley, 273 F.3d 429 (2d Cir. 2001).

115. *See* 17 U.S.C. § 1201(a)(1)(A), (a)(2), (b)(1) (2006).

116. Some of these concerns have been litigated in recent years under the DMCA. *321 Studios*, 307 F. Supp. 2d at 1092–99 (involving trafficking in device capable of circumventing encryption codes placed on DVD movies); *Elcom*, 203 F. Supp. 2d at 1118–19 (involving trafficking in device that could circumvent technological protections applied to eBooks); *Reimerdes*, 111 F. Supp. 2d at 303 (involving trafficking in device that circumvents encryption code placed on DVD movies).

## B. Historical Development of Programming Strategies

Over time, the software industry has matured significantly, impacting the amount of appropriate legal protection necessary for programmers' efforts. In the latter part of the twentieth century, as the demand for stand-alone business applications increased, so too did the need to develop more user-friendly programs. With the business community as software consumers, computer scientists needed to improve code-writing strategies to avoid errors and bugs[117] that could not easily be corrected by average business users. The commercialization of code also brought with it increased need to protect the commercial value of the code through legal means.

### *1. Structured Programming*

One major advance in programming in this context was the development of the modularized "control structure" approach to programming. This approach allowed enhanced performance and error testing, along with decreased development time and maintenance costs.[118] It largely achieved these aims using the idea that programs should be based on three simple control structures: sequence, selection, and repetition.[119] The *sequence* structure in structured programming is the idea that statements in computer source code are executed one after the other in a linear fashion.[120] The *selection* control structure presents a number of processing options in the code, and the option chosen depends on the result of a decision criterion.[121] People who have looked at source code might recognize this as the "IF... THEN" types of commands often seen in such code. The *repetition* control structure, also often referred to as looping, is utilized to execute a particular instruction or instruction set more than once while a particular condition is satisfied.[122] Again, people familiar with source code might recognize this as a "DO... WHILE" or "DO... UNTIL" command.[123]

In the structured programming context, much code will be

117. A computer bug is "an error in a computer program or system." THE OXFORD ENGLISH DICTIONARY 123 (Della Thompson ed., 2d ed. 1996).

118. ROBERT A. SZYMANSKI ET AL., INTRODUCTION TO COMPUTERS AND INFORMATION SYSTEMS 290 (1988).

119. *Id.*

120. *Id.*

121. *Id.*

122. *Id.*

123. Technically, there is also a fourth type of control structure which tended to be used more in early programming than it is today because of its potential to cause problems in execution. This fourth control structure is known as a "GOTO" statement, which allows a program to "jump" to other points in the relevant code. *Id.* Again, those who have looked at source code will probably be familiar with this idea. However, it can cause problems because it can make programs difficult to follow, and interfere with the idea of sequence: that is, that you follow the program code from the top to the bottom in a relatively sequential structure. *Id.*

repetitious, and independent reinvention is likely to be common, along with conscious re-utilization of coding structures. If structured programmers limit their programming methodologies to those that involve sequence, selection, and repetition, it is likely that many programmers will be thinking in the same way and will be reinventing the wheel to a great extent. The idea behind good programming is not necessarily to be particularly original and creative, but to be functional and to reuse similar ideas and structures. This is a bad fit for copyright law's focus on regulating unauthorized copying and reproductions.

### 2. *Modularization*

Another important concept that has been developed in modern programming theory and practice is *modularization*. This involves breaking up sub-tasks within a program into self-contained units or modules, each of which can be dedicated to a single function.[124] This is useful in complex programming because it simplifies the program structure, and enables different programmers to efficiently work on different aspects of the same program.[125] Another benefit of modularization is that once the discrete modules have been written, they can easily be understood, reused within the same or another program, and independently modified if necessary.[126]

The idea of modularization provides another indication that copyright law is not a particularly good fit for modern software products. One of the key features of good programming today is the ability of discrete modules to be reused by different people in different contexts with or without modification.[127] This clearly runs counter to the aims of copyright law, which are to prevent such reproductions without the authority of the original author. Perhaps this is an overbroad statement in the commercial software context. It is likely that corporate software developers would applaud a total ban on a competitor's free use of modules of their code. However, this is where we see a significant disconnect between various areas of the software developing community. In pure computer science terms, the ideas of structured programming and modularization encourage copying. At least in recent years, using someone else's module and building on it to create a new program is what software programming has been about. In fact, the open source movement encourages copying by providing free access to open source code, thus enabling programmers to work on improvements.[128] The

124. ROBERTSON ET AL., *supra* note 68, at 104.

125. *See id.*

126. *Id.*

127. *See id.*

128. *See generally* Dennis M. Kennedy, *A Primer on Open Source Licensing Legal Issues: Copyright, Copyleft and Copyfuture*, 20 ST. LOUIS U. PUB. L. REV. 345 (2001); Marcus Maher, *Open Source Software: The Success of an Alternative Intellectual Property Incentive Paradigm*, 10 FORDHAM

practice of software programming should be better reflected in the legal protections granted to the commercial software community. Because there are now many avenues outside copyright law to effectively protect software code, including patents, modern DRM measures, and restrictive licenses that are clearly enforceable under contract law,[129] there is no need for copyright law to protect code.

### 3. *Object-Oriented Programming*

A final point to make about recent advances in computer programming theory and practice relates to the advent of object-oriented programming. Object-oriented programming further builds on the structured programming approach and is particularly important for the creation of large-scale business applications.[130] It is a technique that relies on creating a set of interacting *objects* rather than a set of interrelated *functions*.[131] In this form of programming, an object refers to a thing we might encounter in the real world, such as a file, document, customer record, or bank account. Objects have various properties including a name, data with attributes having a value at any given point in time, and operations or methods that can be performed on the relevant data.[132] Each object may also be an instance of a *class* of objects.[133] Classes can be progressively created from other classes by *copying* some of the attributes and methods from the parent class.[134] The term "inheritance" is used to describe a new class of objects taking attributes and methods from a parent class.[135]

Without going into unnecessarily complex detail about object-oriented programming, the idea behind this form of program design is for the programmer to create a program that uses objects in the same way that people relate to real world items. For example, just as a bank account may have certain properties and attributes and may be dealt with in a certain way in the real world, a similar functionality can be achieved as part of the program structure. Checking accounts may have certain similarities to credit accounts, such as account numbers, customer names, and transaction records, but there may also be differences, such

INTELL. PROP. MEDIA & ENT. L.J. 619 (2000); David McGowan, *Legal Implications of Open Source Software*, 2001 U. ILL. L. REV. 241; Stephen McJohn, *The Paradoxes of Free Software*, 9 GEO. MASON L. REV. 25 (2000); Miller, *supra* note 67; Christian H. Nadan, *Open Source Licensing: Virus or Virtue?*, 10 TEX. INTELL. PROP. L.J. 349 (2002); Shawn W. Potter, *Opening Up to Open Source*, 6 RICH. J.L. & TECH. 24 (2000); Greg R. Vetter, *"Infectious" Open Source Software: Spreading Incentives or Promoting Resistance?*, 36 RUTGERS L. J. 53 (2004).

129. *See* sources cited *supra* note 96.

130. ROBERTSON ET AL., *supra* note 68, at 175.

131. *See id.*

132. CAPRON & JOHNSON, *supra* note 71, at 460–63; ROBERTSON ET AL., *supra* note 68, at 175.

133. ROBERTSON ET AL., *supra* note 68, at 175.

134. *Id.* at 176.

135. *Id.*

as rules for overdrawing the account. These attributes and methods can become parts of objects within a program and "credit account objects" may be developed as a subset of "checking account objects."

Lawyers, lawmakers, and policymakers must understand that one of the key features of object-oriented design and programming is *reusability*.[136] Objects can be reused within the same and different programs by the same or different programmers. Further, new objects can be created by inheriting, or copying, attributes and methods from parent objects as noted above. Again, these may be utilized within the same program as the parent objects or in different programs written by the same or different programmers. In terms of the copyright paradigm, again we see a poor fit between software code and copyright law. While copyright law is intended to prohibit unauthorized copying of fixed literal expressions of ideas, the object-oriented programming technique relies on substantial copying of structures or exact literal expressions of code within objects.[137]

## C. Programming Methodology and the Legal Protection of Software

The software industry's common reuse of code is not affected by other avenues of legal protection as it is by copyright law. Patents, for example, protect novel[138] and non-obvious[139] inventions.[140] A patent grant will provide protection against anyone who attempts to commercially exploit the function of an invention without the permission of the patent holder.[141] The copying question is irrelevant because the focus of a patent infringement action is on the function of the code, not the code itself. Thus, a competitor could legally copy software code if it resulted in a completely different functional invention. This would be commonplace in the software industry and would arguably, under current principles, be an infringement of copyright. However, it would not necessarily amount to a patent infringement because protecting the nature of an *invention* and protecting specific literal elements of code utilized in an invention are, at least theoretically, two different things.

Additionally, typical incremental developments in programming make it unlikely that many software programs will rise to the level of invention required in the patent sense.[142] Thus, patent law does not give

136. Capron & Johnson, *supra* note 71, at 463.

137. *See, e.g.*, John Edwards, *The Object-Oriented Express*, CIO Mag., Nov. 1, 1995, at 74.

138. 35 U.S.C. §§ 101, 102 (2006).

139. *Id.* § 103.

140. *Id.* § 101.

141. *Id.* § 271.

142. Samuelson et al., *supra* note 13, at 2330–31 ("Innovation in software development is typically incremental. Programmers commonly adopt software design elements—ideas about how to do particular things in software—by looking around for examples or remembering what worked in other

rise to the same over-protection problems as an aggressive application of copyright law, as long as patents are not granted too freely and interpreted too broadly.[143]

In any event, DRM and contractual measures, for example, coupled with trade secrecy, likely provide sufficient stand-alone levels of legal protection for software producers.[144] DRM and contractual measures can provide software developers with relatively inexpensive and immediate protection for their endeavors, thereby counteracting a number of potential concerns with scaling back copyright protection. Also, with recent advances in encryption technologies, it is much easier for software developers to rely on contract and technology to help protect their work against unauthorized access and use than it has been in the past.

In the 1980s and 1990s, for example, it was not clear what kind of role patent, trade secrecy, contract, and DRM measures would ultimately play in the protection of software and its underlying code. Nor was it clear that trade secrecy could work effectively in this context with DRM measures and contractual licenses. Subsequent history has shown that these avenues of protection have certainly proven effective, if not overly protective in some cases.[145] Maintaining copyright protection for the fixed

programs. These elements are sometimes adopted wholesale, but often they are adapted to a new context or set of tasks. In this way, programmers both contribute to and benefit from a cumulative innovation process. While innovation in program design occasionally rises to the level of invention, most often it does not.").

143. Some would argue that this is what has happened in the United States in the past decade, although this view may be disputed. *See, e.g.*, Julie Cohen & Mark Lemley, *Patent Scope and Innovation in the Software Industry*, 89 CAL. L. REV. 1, 3 (2001) (noting the high number of software patents granted in the United States and suggesting new developments in patent policy that could better support software innovation); Arti Rai, *Addressing the Patent Gold Rush: The Role of Deference to PTO Patent Denials*, 2 WASH. U. J.L. & POL'Y 199, 202 (2000) (acknowledging the plethora of high technology patents).

144. *See* ProCD, Inc. v. Zeidenberg, 86 F.3d 1447, 1449 (7th Cir. 1996) (upholding clickwrap and shrinkwrap license applied to computerized database); *In re* RealNetworks, Inc., Privacy Litig., No. 00-C-1366, 2000 WL 631341, at *5 (N.D. Ill. May 8, 2000) (holding that web users had agreed to an arbitration agreement in a clickwrap license); Cohen, *supra* note 96, at 1090 ("A new wind is blowing in copyright law. For centuries, authors and their assignees have relied primarily on federal copyright law to define and protect their legal rights. Suddenly, that may be about to change. New developments in digital technology offer copyright owners the tantalizing possibility of near-absolute control of their creative and informational content, even after its delivery to end users, via self-enforcing digital contracts. Copyright owners, along with purveyors of other (noncopyrightable) informational content, envision using these contracts to secure—and redefine—their 'informational rights.' Within this vision of private ordering and technological self-help, contract law rather than copyright law is paramount. Limits on information ownership set by the public law of copyright are conceived as optional restrictions that can be avoided using appropriate contractual language."); Zynda, *supra* note 96, at 504 (describing how courts have generally upheld the validity of shrinkwrap and clickwrap licenses applied to computer software).

145. This is was argued, for example, in the Eighth Circuit Court of Appeals case of *Blizzard v. BNetD*, which involved restrictive contractual licensing and DRM measures bolstered by provisions of the DMCA. Brief of Defendants-Appellants at 24-39, Blizzard v. BnetD, No. 04-3654 (8th Cir. Jan. 12, 2005). For copies of the court documents in this case, see Electronic Frontier Foundation, Blizzard v.

literal expression of code in light of these other protection mechanisms is now not only unnecessary, but leads to undesirable complexity, as well as potentially misplaced programming incentives.

## III. SOFTWARE CODE AND DIGITAL COPYRIGHT LAW

### A. THE IDEA-EXPRESSION DICHOTOMY

The main problem with copyright protection in the modern world is that it potentially leads to over-protection of code, particularly in concert with other protection mechanisms for software producers. Other than the overly long duration[146] of copyright protection in the software code context, a significant reason for this over-protection lies in the subject-matter that has been protected by copyright law over the last decade or so. Copyright law is only intended to protect original literal expressions of works, and not their underlying ideas.[147] However, software developers have asserted copyrights in the most minimally original modules of code in recent years.[148] Courts, potentially confused about the validity of these copyrights, may be prepared to uphold them.[149] Even in the absence of judicial support for such copyrights, the very existence of a copyright in a program has the potential for a significant chilling effect on innovation in relevant markets. As previously noted, copyrights are cheaper and easier to obtain and to register than patents. In fact, in most jurisdictions outside the United States, copyrights cannot be registered, but are still valid, effective, and enforceable.[150]

Interestingly, the possibility of over-protection of software on copyright subject matter grounds was predicted in the 1980s by Professor Menell, who noted that even CONTU had recognized as early as 1978 that it was going to be impossible to establish a precise line between copyrightable expression and unprotectible underlying processes.[151] Professor Menell argued that the ultimate wisdom of the decision to protect software code through copyright law would depend on judicial interpretations of where the line between expression and underlying

BnetD, http://www.eff.org/IP/Emulation/Blizzard_v_bnetd (last visited Nov. 5, 2006).

146. As noted above, software protection generally lasts for the author's life plus seventy years in most jurisdictions. *See, e.g.*, 17 U.S.C. § 302(a) (2006).

147. This is often referred to as the "idea-expression" dichotomy. *See, e.g.*, Paul I. Kravetz, *"Idea/Expression Dichotomy" and "Method of Operation": Determining Copyright Protection for Computer Programs*, 8 DEPAUL BUS. L.J. 75, 76 (1995).

148. *See, e.g.*, Lexmark Int'l, Inc. v. Static Control Components, Inc., 253 F. Supp. 2d 943 (E.D. Ky. 2003), *vacated*, 387 F.3d 522 (6th Cir. 2004).

149. There was certainly confusion between the lower court and appeals court in the *Lexmark* litigation on this point, with the lower court assuming the relevant lock out code was copyrightable, 253 F. Supp. 2d at 958, and the appeals court disputing this finding, 387 F.3d at 529.

150. *See* LEAFFER, *supra* note 4, § 7.1.

151. Menell, *supra* note 14, at 1047.

processes was drawn.[152] Drawing the line too far in favor of copyright protection would lead to undesirable software monopolies and would likely chill competing innovations, whereas drawing the line in the other direction would provide insufficient protection for initial software innovations and would discourage many advances in relevant fields.[153] However, it is not only judicial line drawing that has been implicated in the pattern of over-protection of software. The mere fact that a copyright claim is made with respect to a particular program can discourage innovation, particularly if the claim is made by a powerful corporation with the wherewithal to threaten small-scale competitors with protracted and expensive copyright litigation.

As long as software copyrighting requires so few costs and formalities, this specter is very real. Thus, what the software industry perceived in the 1980s as advantages to the use of copyright law, in terms of costs and formal requirements for copyrighting code, now appear to be factors that potentially lead to chilling of innovations. As software programming methodology develops, the increased reliance on concepts like modularization and object-oriented programming techniques add to this problem. This is because developments in the industry are increasingly likely to rely on duplication of existing modules and ideas. Additionally, as programs are more and more modularized, each module may be less and less original in the copyright sense.

It is promising that some federal circuit judges have recently begun to question the copyrightability of certain computer code with respect to the idea-expression dichotomy. For example, in the appeal decisions in *Lexmark International, Inc. v. Static Control Components*[154] and *Chamberlain v. Skylink*,[155] some judges expressed concern that too much code was being copyrighted without any critical evaluation of its copyrightability. Both of these cases involved "lock out" codes.[156] Lock out codes are modules of software code that operate as security systems attached to a device, such as a computer, printer, or an automobile, to bar the use of unauthorized components with the product.[157] The majority in *Lexmark* noted that such lock out codes generally fall on the functional idea side of the idea-expression dichotomy, rather than on the original expression side.[158] Thus, they should not often be granted copyright protection, either because they are not sufficiently original to

152. *Id.* at 1048.
153. *Id.* at 1047–48.
154. 387 F.3d at 542–43.
155. 381 F.3d 1178, 1201 (Fed. Cir. 2004).
156. *Lexmark*, 387 F.3d at 543; *Chamberlain*, 381 F.3d at 1183.
157. *Lexmark*, 387 F.3d at 536.
158. *Id.*

attract copyright protection[159] or because the idea is merged with the expression, effectively barring copyright protection.[160]

A number of fundamental problems inherent in modern applications of copyright law still prevent courts from drawing clear lines between copyrightable and un-copyrightable code. Although the majority in the *Lexmark* appeal was arguably on the right track in terms of their impulse to limit copyright protection for code where the idea and expression appear to be merged, in many ways the decision left open more questions than it answered. For one thing, it is simply not clear on the current state of the law, even as interpreted by the Sixth Circuit in *Lexmark*, how and when the originality and merger questions should be asked in a software code case. Both the *Lexmark* majority and Judge Feikens, in his partially concurring opinion, noted that there appears to be judicial dissonance regarding when to ask questions relating to the merger of idea and expression.[161] Does the merger doctrine operate as an initial bar to copyrightability, or does it only come into play in answering the substantial similarity question in an infringement proceeding by way of a defense?[162]

This is an important question in the computer code context because copyrighting code per se, even outside the litigation context, may serve as an effective bar to innovative competition in the software industry. A competitor may be wary of arguing a merger or *scènes à faire* defense in an infringement proceeding, particularly where she has less financial resources to hire independent experts than the initial copyright holder. It might be much more useful for the continued development of the software industry if there were some ex ante test for copyrightability of code based on originality and merger grounds. At least in such a scenario, the burden of proof of originality could shift to the plaintiff.

Even if an attempt were made to shift the burden of proof to the defendant, the question would arise as to how to develop appropriate guidelines to determine copyrightability on originality and merger grounds. Easily quantifiable factors such as the length of a particular program could not serve as determinative guidelines here. As noted in *Lexmark*, brief programs may reveal high levels of creativity in their expression.[163] Conversely, lengthy programs consisting of standard software modules may not reveal particularly high levels of creativity. Furthermore, even if some effective guidelines could be developed for determining originality, the application of these guidelines may create

159. *Id.* at 536–37 (describing the "originality" criterion for copyright law as set out by the Supreme Court in the *Feist* case).

160. *Id.* at 536 (describing the basic merger problem with respect to lock out codes).

161. *Id.* at 535; *id.* at 556–59 (Feikens, J., concurring in part and dissenting in part).

162. *Id.* at 538–39 (majority opinion).

163. *Id.* at 542.

perverse incentives in programming.

If, for example, code needed to be expressed in a more convoluted manner to pass the originality test for copyrightability, programmers may waste time on writing code with more expressive bells and whistles that does not function any more efficiently, or that perhaps runs less efficiently, than a less "original" alternative. Again, this evidences how poor a fit copyright law is for code. Copyright focuses on original *expression* while code is about efficient functioning. Even an approach that applies the originality standards more rigorously to code than is currently the case would run the risk of creating all the wrong incentives in the software industry.

There may be some cases in which the efficiency concerns are so narrow that there are only one or two options for writing certain code,[164] and the copyrightability question might be more easily answered in these cases. As suggested in the *Lexmark* case, this may be so with respect to many lock out codes.[165] However, the ease with which these few cases were handled does not solve the chief problem. With programming methodology moving toward modularization and object-oriented concepts, it is counterintuitive to continue relying on a legal protection that is a poor fit for the realities of the industry, and that potentially emphasizes the wrong incentives.

Because copyright protects the wrong aspects of software, its literal expression rather than its underlying utility, courts have been faced with the dilemma identified by Professor Menell many years ago: judges have to either interpret copyright laws too broadly and risk over-protection, or too narrowly and risk under-protection.[166] This may have been regarded as a necessary evil at the time Congress adopted the policy of favoring copyright protection for code, but it is no longer necessary in light of subsequent developments in law and technology. Trade secret law, augmented by modern DRM measures and contractual licensing can now provide effective protection for software in ways not previously possible. Patents should also have some role to play here, not in protecting specific iterations of code, but in protecting software related inventions that truly are novel and non-obvious.[167] Taking copyright out of the picture and re-focusing on these alternative protection avenues for software would be preferable to straining the boundaries of copyright law.

If copyright law is retained with respect to software code, significantly more work needs to be done to resolve the originality and merger questions. This work should not be left to the judiciary when a

164. *See, e.g.*, Computer Assocs. Int'l v. Altai, Inc., 982 F.2d 693, 708 (2d Cir. 1992).
165. *Lexmark*, 387 F.3d at 536.
166. Menell, *supra* note 14, at 1047–48.
167. *See* Cohen & Lemley, *supra* note 143.

dispute arises. It should be dealt with by Congress or the Copyright Office, and should be informed by experts in the computer science field. This approach may look more broadly at developments in computer programming methodology, rather than focus on specific issues arising in particular cases. Additionally, an ex ante approach to originality and merger could be developed as a result of such an inquiry. This could avoid the chilling effect created when software developers can register virtually any code at the Copyright Office.

## B. Digital Rights Management and the DMCA

A more recent development in twenty-first century copyright law that affects the software copyright question is the enactment of the anti-circumvention and anti-trafficking provisions of the DMCA.[168] These provisions are intended to bolster DRM measures applied to digital copyright works to control unauthorized access to and use of those works.[169] The relevant provisions of the DMCA create civil and criminal liability for circumventing an access-control measure,[170] and for trafficking a device that can circumvent an access or copy control measure.[171] These provisions have generated some criticism, particularly with respect to their impact on otherwise legitimate uses of digital copyright works, such as fair use.[172] However, their aim has been to stem

168. 17 U.S.C. § 1201 (2006).

169. *See, e.g.*, C.J. Alice Chen & Aaron Burstein, *The Law and Technology of Digital Rights Management: Forward*, 18 Berkeley Tech. L.J. 487, 489 (2003) ("[Digital Rights Management] systems fall within the category of 'technological protection measures' that the Digital Millennium Copyright Act (DMCA) explicitly protects against circumvention. Congress' stated goal, in granting copyright holders a right of access to their works, was to promote the emerging role of digital technology in commerce without 'affect[ing] rights, remedies, limitations, or defenses to copyright infringement, including fair use.' The DMCA itself, however, does not prescribe any technological safeguards of these rights, remedies, limitations, and defenses, nor does it excuse users who circumvent access controls to make lawful use of copyrighted content. The DMCA thus grants copyright holders broad leeway to encode access and usage policies into DRM systems that may effect a balance of rights quite different from the one found within traditional copyright law." (citations omitted)).

170. 17 U.S.C. § 1201(a)(1)(A). There is no sanction on circumventing a *copy control measure* (compare to an access control measure). This is a legislative attempt, along with § 1201(c)(1), to preserve fair use under the DMCA. For a more detailed discussion of the limitations of fair use in this context, see Lipton, *supra* note 59, at 531–36; R. Anthony Reese, *Will Merging Access Controls and Rights Controls Undermine the Structure of Anticircumvention Law?* 18 Berkeley Tech L.J. 619 (2003); Pamela Samuelson, *Intellectual Property in the Digital Economy: Why the Anti-Circumvention Regulations Need to be Revised*, 14 Berkeley Tech. L.J. 519, 548–54 (1999) (providing examples of the relationship between trafficking in circumvention devices and circumvention conduct per se in cases where the circumventor does not necessarily have the technological means to create a circumvention device to make a fair use of a relevant work); *id.* at 557 (suggesting that the anti-trafficking provisions of the DMCA should be amended to preserve the ability to manufacture a circumvention device for legitimate uses, including fair use).

171. 17 U.S.C. § 1201(a)(2), (b)(1).

172. *See, e.g.*, Jacqueline D. Lipton, *Solving the Digital Piracy Puzzle; Disaggregating Fair Use from the DMCA's Anti-Device Provisions*, 19 Harv. J.L. & Tech. 111 (2005); Lipton, *supra* note 59, at 531–36; Reese, *supra* note 170; Samuelson, *supra* note 170, at 548–54 (providing examples of the

the tide of digital copyright piracy,[173] particularly with respect to digital works such as eBooks,[174] digital music, and digital movies.[175] This seems reasonable in light of the fact that the same technology that enables digital content industries to quickly and cheaply distribute high quality digital copies of their works to consumers also enables digital copyright pirates to do the same thing.[176]

However, some problems arise when these provisions of the DMCA are utilized to protect copyrighted software code. Putting to one side the question whether copyright effectively protects the valuable elements of software code, there is the additional problem that software code is incidentally incorporated into physical products such as automotive parts, audio-visual equipment, computer hardware, and even standard electrical appliances.[177] Some cases have arisen recently in which manufacturers of physical goods incorporating software code have attempted to utilize the DMCA to restrict competition for replacement goods in a downstream market.[178] This is an issue that does not arise with other kinds of digital copyright works because they are not functional in the same way that software code is functional. Unlike software code, digital music, movies, and books are basically digitally stored analogues of traditional versions of those works stored in paper, celluloid, or magnetic tape formats. Computer software code is utilitarian and does not represent an artistic or literary endeavor in the same way as these other works. One is much more likely today to find computer code within various tangible utilitarian products such as garage door

---

relationship between trafficking in circumvention devices and circumvention conduct per se in cases where the circumventor does not necessarily have the technological means to create a circumvention device to make a fair use of a relevant work); *id.* at 557 (suggesting that the anti-trafficking provisions of the DMCA should be amended to preserve the ability to manufacture a circumvention device for legitimate uses, including fair use).

173. ELECTRONIC FRONTIER FOUNDATION, *supra* note 60, at 1 (describing that the aim of the DMCA was to prevent circumvention of digital piracy protections attached to copyright works); Burk, *supra* note 53, at 1135 (noting that the legislative aims behind these provisions were to prevent piracy in digital copyright works).

174. United States v. Elcom Ltd., 203 F. Supp. 2d 1111, 1118–19 (N.D. Cal. 2002) (involving trafficking in a device that could circumvent technological protections applied to eBooks).

175. 321 Studios v. Metro Goldwyn Mayer Studios, Inc., 307 F. Supp. 2d 1085, 1092–99 (N.D. Cal. 2004) (involving trafficking in a device capable of circumventing encryption codes placed on DVD movies); Universal City Studios, Inc. v. Reimerdes, 111 F. Supp. 2d 294, 322–24 (S.D.N.Y. 2000), *aff'd sub nom.* Universal City Studios v. Corley, 273 F.3d 429 (2d Cir. 2001) (involving trafficking in a device that circumvents encryption code placed on DVD movies).

176. A similar point was made recently by the Supreme Court in the digital file-sharing context. Metro-Goldwyn-Mayer Studios, Inc. v. Grokster, Ltd., 125 S.Ct. 2764, 2775 (2005) (noting that digital distribution mechanisms enable identical copies of a protected work to be made very easily).

177. *See, e.g.*, UNIF. COMPUTER INFO. TRANSACTIONS ACT § 102 cmt. 7 (amended 2002), 7 U.L.A. pt. II, at 204 (2002) (noting that even toasters may contain microprocessors in modern commerce).

178. *See, e.g.*, Chamberlain Group, Inc. v. Skylink Techs., Inc., 292 F. Supp. 2d 1023, 1046 (N.D. Ill. 2003), *aff'd*, 381 F.3d 1178 (Fed. Cir. 2004); Lexmark Int'l, Inc. v. Static Control Components, Inc., 253 F. Supp. 2d 943, 946–47 (E.D. Ky. 2003), *vacated*, 387 F.3d 522 (6th Cir. 2004).

openers,[179] laser printers, and associated printer toner cartridges[180] than in the past.

Manufacturers of these devices have argued that where they have used a DRM measure to tie a replacement part, like a printer toner cartridge, to a particular physical good, like a printer, they should be able to avail themselves of the DMCA to prevent unauthorized access to, or use of, code stored within either or both devices. In the *Lexmark* litigation, for example, Lexmark had allegedly inserted copyrighted computer code both within its toner cartridges and associated printers.[181] One of the functions of the code in the cartridges was to ensure that only authorized Lexmark cartridges could be operated within particular Lexmark printers.[182] This was achieved by utilizing two computer programs, a Toner Loading Program (TLP) inserted into printer toner cartridges, and a Printer Engine Program (PEP) located on the hard drive of the relevant computer printers.[183] The TLP operated as a lock out code to ensure that no unauthorized cartridges would work within a relevant printer.[184] It did this by exchanging data with the PEP, so the PEP could authenticate the authorized cartridge before anything could be printed using the cartridge.[185]

To date, Lexmark has failed both in copyright infringement claims against a competitor, Static Control Components (SCC), and in DMCA claims. The copyright claims failed because the Sixth Circuit Court of Appeals was not convinced that the TLP, the code within the cartridges that SCC copied, was copyrightable.[186] Either the idea had merged with the expression[187] or the code was insufficiently original to attract copyright protection.[188] The DMCA claims failed because the appellate court was not convinced that the lock out code in the printer toner cartridge was an effective technological protection measure required for a successful action.[189] Even though the TLP housed in the Lexmark cartridges functioned as a lock out code to prevent the operation of the printer with an unauthorized cartridge inserted, the allegedly protected copyright work, a PEP residing inside the relevant printers, was accessible in other ways. Anyone who lawfully purchased a relevant

179. *Chamberlain*, 292 F. Supp. 2d at 1046.
180. *Lexmark*, 253 F. Supp. 2d at 946.
181. Lexmark Int'l, Inc. v. Static Control Components, Inc., 387 F.3d 522, 529–30 (6th Cir. 2004).
182. *Id.* at 530.
183. *Id.* at 529–30.
184. *Id.* at 530.
185. *Id.* For a more detailed discussion of this point, see Lipton, *supra* note 59, at 499–501.
186. *Lexmark*, 387 F.3d at 534–37.
187. *Id.* at 534–36.
188. *Id.* at 536–37.
189. *Id.* at 546–49.

printer could access the PEP code within the printer.[190]

However, the problem does not end with Lexmark. It would be very simple for a manufacturer to get around the reasoning in the *Lexmark* decision by better protecting the relevant computer code from unauthorized access or use. If the PEP had been properly encrypted in *Lexmark*, the result may have been very different. If, for example, purchasers of Lexmark printers were unable to access, read, copy, or distribute the PEP from the hard drive of the printer due to additional encryption applied within the printer hardware itself, the PEP may have been protected by an effective technological protection measure as required by the legislation.[191]

In such a case, if a competitor such as SCC had attempted to circumvent the technological protection in the creation of an interoperable competing toner cartridge, a DMCA claim may have succeeded. This point was made forcefully by Judge Merritt in his concurring opinion in *Lexmark*:

> We should make clear that in the future companies like Lexmark cannot use the DMCA in conjunction with copyright law to create monopolies of manufactured goods for themselves just by tweaking the facts of this case: by, for example, creating a Toner Loading Program that is more complex and "creative" than the one here, or by cutting off other access to the Printer Engine Program.[192]

Judge Merritt's concerns were twofold. First, he was concerned about the reach of traditional copyright law and its potential to protect relatively unoriginal code like the basic lock out program.[193] Second, he was worried about the potential for manufacturers of physical goods to artificially lock up computer code incorporated into those products in order to run DMCA arguments to prevent downstream competition in after-markets for replacement parts.[194] Overall, he was worried that the reasoning of the majority did not go far enough toward preventing these outcomes.[195] In this vein, he advocated that the DMCA should be interpreted in the future in a manner that makes it clear that the statute requires plaintiffs to show a purpose to pirate the defendant's copyright work.[196] Although this is clearly a laudable suggestion, it is not what the

190. *Id.* at 546 ("Anyone who buys a Lexmark printer may read the literal code of the Printer Engine Program directly from the printer memory, with or without the benefit of the authentication sequence, and the data from the program may be translated into readable source code after which copies may be freely distributed.").

191. Lipton, *supra* note 59, at 506.

192. *Lexmark*, 387 F.3d at 551 (Merritt, J., concurring).

193. *Id.* at 551–52.

194. *Id.* at 552.

195. *Id.* at 551–52.

196. *Id.* at 552–53. For alternative approaches to this problem, see generally Burk, *supra* note 53; Lipton, *supra* note 59.

statute currently says and there is no requirement for future courts to interpret it in this way.

Obviously, as with traditional copyright infringement actions, there are some statutory and common law defenses available to those accused of DMCA infringement in such cases. The defendant in *Lexmark* argued fair use[197] and also suggested that the plaintiff had engaged in copyright misuse.[198] These defenses were not discussed in great detail by the appeals court as the majority was not convinced that Lexmark could successfully sustain its copyright and DMCA infringement actions in any event.[199] SCC also raised the statutory interoperability defense from the DMCA, but again this was not pursued in detail by the judges.[200]

In any event, reliance on these defenses in such cases raises the same problem as asking defendants to rely on similar defenses in traditional copyright infringement actions involving computer code. It puts the burdens of proof in the wrong places and potentially stifles innovation in competing markets, particularly in markets for physical goods that are only tangentially associated with copyrightable subject matter. It may be far better to bar copyright and DMCA actions altogether in these kinds of cases on the basis that the cases are not really about protecting copyright works. Alternatively, as suggested by Judge Merritt, even if the relevant code is potentially copyrightable, there should be at least a presumption against a DMCA action unless the plaintiff can establish a clear purpose to pirate on the part of the defendant.[201]

Taking this course and interpreting the DMCA in the way suggested by Judge Merritt, or alternatively amending the legislation to incorporate his presumption[202] may well resolve this problem in practice, but it does not deal with the problem at its foundation. The root of this particular problem is the copyrightability of software code per se. If code were not copyrightable, it would not be possible for manufacturers of physical goods to incorporate functional code within those goods and then claim both copyright and DMCA infringements against competitors trying to compete in after-markets for replacement parts.

197. *See* 387 F.3d at 544 (Merritt, J., concurring) ("In view of our conclusion on this preliminary-injunction record that the Toner Loading Program is not copyrightable, we need not consider SCC's fair-use defense.").

198. Lexmark Int'l, Inc. v. Static Control Components, Inc., 253 F. Supp. 2d 943, 965–66 (E.D. Ky. 2003), *vacated*, 387 F.3d 522 (6th Cir. 2004). The copyright misuse defense was not discussed in the appeal judgment.

199. *Lexmark*, 387 F.3d at 550–51.

200. *Id.* at 551.

201. *Id.* at 552 (Merritt, J., concurring) ("[A] better reading of the statute is that it requires plaintiffs as part of their burden of pleading and persuasion to show a purpose to pirate on the part of defendants.").

202. For a similar suggestion regarding a legislative presumption that protection of a copyright work is a central commercial concern of the plaintiff, see Lipton, *supra* note 59, at 521–28.

Both copyright law and the DMCA can potentially operate effectively to protect digital copyright works that are truly digital analogues of existing creative works. However, prior to the digital revolution, this legislation did not work well with respect to software code. Code is utilitarian in character and its very functionality not only makes copyright law a bad fit for protecting it, but also creates unintended negative consequences as copyright law matures in the digital age to protect other forms of digital copyright works against unauthorized access and use. It was not the intention of the drafters of the DMCA to allow manufacturers of printers and toner cartridges to utilize the anti-circumvention provisions to impinge on unwanted competition in downstream after-market replacement parts.[203] However, the copyrightability of software code potentially leads to such outcomes.

Although there are clearly a variety of quick fixes to this practical problem, such as judicial or legislative presumptions against the operation of the DMCA in these cases,[204] it is more logically satisfying and practically appropriate to deal with this problem at its heart. In other words, it is the copyrightability of code per se that creates the potential problem of utilizing the DMCA to prevent competition in downstream markets for tangible goods incorporating code. Even though code resembles a traditional copyright work because it is written in a fixed literal format, it is clearly not traditional copyright subject matter because its value is not in its expression.[205] Eliminating, or significantly scaling back, copyright protection for code could curtail problems that arise as copyright law becomes more stringent in its protection of other forms of digital works, such as digital music, movies, and literature.

## IV. Alternatives to Copyright Protection

### A. Trade Secrecy in the Twenty-First Century

This leads us to the question of whether copyright protection for computer software code should continue at all. Removing copyright protection altogether for code would clearly be a radical move. The open source movement relies heavily on its ability to copyright code and to utilize restrictive licensing provisions, such as the General Public License (GPL)[206] to protect the open nature of the code. As Professor Ginsburg

203. Burk, *supra* note 53, at 1135.

204. *Lexmark*, 387 F.3d at 552–53 (suggesting judicial imposition of "purpose to pirate" presumption where plaintiff must demonstrate purpose to pirate on part of defendant before burden of proof shifts to defendant to raise available statutory defenses); Burk, *supra* note 53 (discussing the development of "anti-circumvention misuse" defense); Lipton, *supra* note 59, at 521–28 (suggesting legislative imposition of presumption that plaintiff must demonstrate that protection of copyright work is a substantial commercial concern in the matter before burden shifts to defendant).

205. Menell, *supra* note 14, at 1047–48.

206. The GPL is probably the most well known form of open source license. *See* Miller, *supra* note

noted many years ago, copyright is also accepted as an international medium for code protection.[207] Therefore it is not just a question of what the United States will do, but also of how that will mesh with the law in other countries.

On the other hand, just because something is widely adopted and accepted does not mean that it should be retained, particularly if it is causing negative practical consequences. Additionally, there is now the likelihood that software is effectively protected in most cases without copyright. We may simply be so used to assuming the existence of, and need for, copyright protection, that we have failed to notice that other measures now provide software manufacturers with more than enough protection.

When Congress decided to encourage copyright protection for software, it was not clear what kind of role patent law, trade secret law, contract law, and DRM measures would play in the protection of valuable software. That picture is much clearer today. Arguably, patents have been over-utilized in relation to software-related innovations.[208] There has certainly proven to be little difficulty, in the United States at least, in patenting software-related inventions.[209] Of course, patents do not protect the literal expression of a programmer's idea, nor do they protect software related inventions that are not sufficiently innovative. They only protect software as part of a novel and non-obvious invention. It is arguable that there is still a need for some protection from those who might duplicate code, utilize it, and distribute it without the authorization of the original developer.

However, the question remains as to whether copyright protection is the answer here. Clearly copyright law is primarily concerned with preventing unauthorized duplications and disseminations of a fixed literal work. With respect to computer code, the same, and perhaps even more effective, protections can be achieved through the use of trade secrecy augmented by DRM measures and contractual licenses. Legislation similar to the Computer Fraud and Abuse Act (CFAA) is also a viable alternative.[210] In the modern world, these avenues of

67, at 496. It is the "general public license" originally promulgated by the GNU project. *Id.* at 496–97.

207. Ginsburg, *supra* note 15, at 2559–60.

208. The author is therefore not arguing that patents are a "silver bullet" to answer all the problems in relation to the potential over-protection of software by intellectual property law. However, the effectiveness—or over-effectiveness—of patents for software protection does suggest that copyrights are an unnecessary addition to the intellectual property matrix for software and that copyrights could easily be scaled back or eliminated even if additional limitations are also required for patent law.

209. Cohen & Lemley, *supra* note 143; Rai, *supra* note 143; World Intellectual Property Organization, *supra* note 95 ("Today, there are an increasing number of software and business methods which are protected by patents in the United States . . . .").

210. 18 U.S.C. § 1030 (2006). The CFAA prevents unauthorized access to computer systems and

protection would likely fill in any gaps left by patent protection and, when compared with copyright protection, more appropriately focus upon the valuable functional elements of software code.

In the early days of copyright protection for software code, trade secrecy was thought to be problematic in the code context because of the ease with which trade secret status could be lost.[211] In the days when DRM measures were not particularly sophisticated and, in many cases, were non-existent, and the enforceability of restrictive contractual licenses was unclear, trade secrecy did not seem an appropriate avenue of protection for code. However, if we now assume that a software developer who develops commercially valuable code will not disseminate it without first encrypting it and imposing easily enforceable licensing provisions, trade secrecy may be more useful than we thought in the 1980s and 1990s.

If computer software code can be effectively kept secret by utilizing DRM measures and restrictive contractual provisions, it clearly meets the definition of trade secret in the Uniform Trade Secrets Act (UTSA) and thus can be protected against unauthorized misappropriation under that legislation.[212] The UTSA definition covers "information, including a formula, pattern, compilation, *program*, device, method, technique, or process"[213] if the information "derives independent economic value, actual or potential, from not being generally known to, and not being readily ascertainable *by proper means* by, other persons who can obtain economic value from its disclosure or use."[214] It also requires that the information be "the subject of efforts that are reasonable under the circumstances to maintain its secrecy."[215] A similar definition of trade secret appears in the federal Economic Espionage Act of 1996 (EEA),[216] which imposes criminal liability for certain trade secret misappropriations.[217]

In the 1980s and early 1990s it may have been very difficult for software manufacturers to effectively, not to mention cost-effectively, impose DRM measures to prevent a program from being readily ascertainable by others. However, the balance has shifted in the last decade or so. If a relatively sophisticated DRM measure augmented by a clearly restrictive contractual license is attached to a program, it should

unauthorized transmission of data within those systems. *See infra* Part IV.B.

211. Menell, *supra* note 14, at 1072.

212. UNIF. TRADE SECRETS ACT § 2 (2005) (injunctive relief for misappropriation of a trade secret); *id.* § 3, 14 U.L.A. 633 (damages for misappropriation of a trade secret).

213. *Id.* § 1(4) (emphasis added).

214. *Id.* § 1(4)(i) (emphasis added).

215. *Id.* § 1(4)(ii).

216. 18 U.S.C. § 1839(3) (2006).

217. *Id.* §§ 1831–1832.

provide a significant level of practical protection to a software manufacturer. Additionally, it should readily satisfy the trade secret definitional requirement of taking reasonable efforts to maintain the secrecy of valuable information. Reliance on trade secrecy augmented by DRM and contractual measures also focuses more realistically on what is economically valuable about software, the functional element of the relevant information, rather than its literal expression.

Trade secrecy perhaps has the perceived disadvantage that some measure of reverse engineering of trade secrets has generally been tolerated, provided that the original item containing the protected information was legally obtained, and the reverse engineering is not in breach of contract.[218] However, with improved DRM and contractual protections in the software context, this should be no more of a problem in the software code context than in relation to any other type of trade secret. In any event, as previously noted, reverse engineering of copyright-protected software code has also been tolerated under copyright law, either as an example of a permitted fair use of a copyright work[219] or through black box techniques that do not technically involve copying.[220] Thus, trade secrets are no more problematic than copyrights in this context.

Regarding the possibility that digital copyright law, particularly in the guise of the DMCA, has the added advantage of protecting a software manufacturer against unauthorized *access* and *use* of an encrypted program, today's trade secret laws may provide the answer, given that secrecy can be adequately protected in practice. The UTSA and EEA are aimed at preventing and imposing civil and criminal sanctions on the failure to prevent, unauthorized access to and use of protected information. The UTSA, for example, prohibits unauthorized misappropriation of a trade secret.[221] In this context, misappropriation is defined to include both unauthorized acquisition of a trade secret,[222] which is presumably the same thing as unauthorized *access* to the relevant information, and unauthorized disclosure or *use* of a trade secret.[223] The criminal sanctions for trade secret theft in the EEA are similarly broad.[224] If these provisions cover the same things as the anti-circumvention provisions of the DMCA in the copyright context, but with a focus on protecting utilitarian aspects of information, is there any

218. 1 ROGER M. MILGRIM, MILGRIM ON TRADE SECRETS § 1.05[2] (2004).

219. Sega Enters. Ltd. v. Accolade, Inc., 977 F.2d 1510, 1520–28 (9th Cir. 1993); LEAFFER, *supra* note 4, § 10.13; *see also* LEMLEY ET AL., *supra* note 34, at 136–38 .

220. Samuelson et al., *supra* note 13, at 2317–18.

221. UNIF. TRADE SECRETS ACT § 2(a), 14 U.L.A. 619 (2005).

222. *Id.* § 1(2)(i), 14 U.L.A. 538.

223. *Id.* § 1(2)(ii) (emphasis added).

224. 18 U.S.C. §§ 1831–1832 (2006).

reason to retain incidental copyright protection for software to prevent unauthorized access and use of code under the DMCA? Probably not, particularly given the demonstrable practical problems this can create.

One could argue that trade secrecy does not help the open source movement because the participants do not wish to retain secrecy in their code and therefore cannot use trade secret laws as an alternative to copyright protection. The answer is obvious here. It is simply necessary to look at the reason why the open source movement relies on copyright law. Copyright is not intended to protect the commercial value of software, but rather to serve an identification function for code in the open source context. Copyright basically describes or *identifies* the property right to which a relevant open source license will attach. Thus, copyright works in concert with contract to ensure that code is *not* over-propertized in the open source context.

Creating an intellectual property right to prevent over-propertization or over-commercialization of a particular item is not the aim of copyright law in the context of software code. If the aim of the open source movement is to use copyright to establish the boundaries of a particular item of intellectual property subject to an open source license, it should be possible to substitute another kind of identification mechanism in this context. In fact, it is questionable whether the open source movement needs any kind of specific legislative intellectual property right to achieve these ends. Would it not be possible for the participants simply to identify the relevant code in a given contractual license without the need for a separate, stand-alone property right to identify the asset in question? It is not immediately clear what copyright gives the open source movement that a well drafted stand alone license would not give to the movement, other than a shorthand way to refer to a particular program. Perhaps the ability to register an open source program on the Copyright Register is helpful in terms of identifying ownership of the relevant rights. Nevertheless, it is not essential in this context. In fact, most jurisdictions outside the United States do not have a copyright register and so cannot rely on copyright registration as evidence of ownership of a copyright work, even in the open source context.[225]

It is arguable that the open source movement does need some kind of intellectual property right on top of contractual licensing provisions to cover situations where it is difficult to establish contracting parties, and there is a concern that someone has used the code inappropriately. An example would be where a proprietary software manufacturer has found some open source code on the Internet, copied it, and propertized it within a commercial product without ever having seen the contractual

225. *See* LEAFFER, *supra* note 4, § 7.1.

license. This might happen if the code is accidentally distributed or posted on a website without the relevant contractual provisions. This might be a situation where either a special subset of copyright should be retained, or, even more appropriately, where a new sui generis right might be developed that meets the needs of the open-source community. Such a new right might encompass both the identifactory functions noted above and an anti-commercialization right enforceable against any downstream user of the code regardless of whether she can be identified as a contracting party with respect to the code. Such a right would effectively enable licensing terms to run with the property right and bind anyone who used or improved the code. It would be an example of using a property (or quasi-property) right to *avoid* undesirable commercialization or monopolization of code.[226]

## B. The Computer Fraud and Abuse Act and Computer Trespass

Other legal protection avenues for software that potentially augment the provisions of the UTSA and EEA might also include the CFAA. Although the CFAA deals mainly with fraudulent activities relating to computers, it is drafted in relatively broad terms and, depending on how the legislation is judicially interpreted, may well cover unauthorized access to, and use of, computer software, at least in some circumstances.

The provisions of the CFAA most likely to be relevant in this context are 18 U.S.C. § 1030(a)(2)(C), (a)(4) and (a)(5). These are all offenses that deal with combinations of unauthorized access to a computer and unauthorized transmission of data from a computer system that cause damage or further a fraudulent activity. The legislation is fairly vague as to the meanings of fraud and damage in this context, although financial damage is contemplated in relation to some of the offenses set out in the legislation.[227] The basic definition of damage is somewhat broader. Damage under the CFAA generally means: "any impairment to the integrity or availability of data, a program, a system, or information."[228] This may or may not include commercial damage suffered by a software developer as a result of a misappropriation of code by a competitor, depending on whether a term such as integrity is interpreted as including integrity of financial value or rather is relegated merely to the physical integrity of the code.

226. On the ability to use property or quasi-property rights to *avoid* monopolies in information property, see generally Michael Carrier, *Cabining Intellectual Property Through a Property Paradigm*, 54 DUKE L.J. 1 (2004); Jacqueline Lipton, *Information Property: Rights and Responsibilities*, 56 FLA. L. REV. 135 (2004).

227. For the purposes of the offenses relating to unauthorized access to a protected computer or unauthorized transmission of data from a computer that cause damage in 18 U.S.C. § 1030(a)(5)(A)(i)–(iii) (2006), "damage" is defined in § 1030(a)(5)(B)(i) to include: "loss to 1 or more persons during any 1-year period . . . aggregating at least $5,000 in value."

228. *Id.* § 1030(e)(8).

Similar points may be made about the potential application of the developing area of computer trespass[229] to the protection of software code. Most of the computer trespass cases to date,[230] such as the CFAA cases, have focused on unauthorized access to computer systems either to access specific data[231] or, in one recent case, to distribute information within an organization.[232] However, in the absence of copyright protection, the CFAA and the evolving computer trespass doctrines may help software developers protect the integrity of their valuable code.

In any event, as interesting as this idea may be in theory, it is probably not necessary in practice because, as previously noted, the law of trade secrets, augmented by enforceable contractual licenses and effective DRM measures, should be able to do the work currently done by copyright law with respect to the legal protection of software code. Trade secrecy, bolstered in this way by contract and DRM, should actually do a better job here than copyright law because it protects the valuable functional aspects of the code and not merely its literal expression. It also does not run into difficult questions of where to draw the line between protected expression and unprotectible ideas. Trade secrecy law will likely protect the actual ideas and information in question provided that reasonable efforts have been made through effective DRM and contractual measures to maintain the secrecy of the relevant code.

## Conclusion

In the 1980s and early 1990s there was much debate and disagreement about the appropriateness of relying on copyright law to protect valuable commercial interests in software code. As history has shown, Congress resolved the matter in favor of copyright protection. However, the reasons for this decision no longer exist. Copyright law is a poor fit for the needs of software developers. Additionally, because of its lack of cost, significant formalities, and its lengthy duration, the specter of over-protection of software code is raised by copyright law.[233]

229. *See generally* Dan L. Burk, *The Trouble with Trespass*, 4 J. SMALL & EMERGING BUS. L. 27 (2000).

230. *See* eBay, Inc. v. Bidder's Edge, Inc., 100 F. Supp. 2d 1058, 1069–70 (N.D. Cal. 2000) (raising "chattel trespass" arguments in relation to computer systems based on personal property rights in servers); CompuServe Inc. v. Cyber Promotions, Inc., 962 F. Supp. 1015, 1021 (S.D. Ohio 1997) ("[E]lectronic signals generated and sent by computer have been held to be sufficiently physically tangible to support a trespass cause of action.").

231. *See In re* Doubleclick Inc. Privacy Litig., 154 F. Supp. 2d 497, 503 (S.D.N.Y. 2001); Shurgard Storage Ctrs., Inc. v. Safeguard Self Storage, Inc., 119 F. Supp. 2d 1121, 1124 (W.D. Wash. 2000).

232. Intel Corp. v. Hamidi, 71 P.3d 296, 299 (Cal. 2003) (involving spam emails sent by ex-employee to current employees of a corporation).

233. The specter of over-protection is also potentially raised by patent and trade secrecy. However, at least these protection methods require somewhat more effort on the part of the code owner over and above creating the code per se.

Although copyright is only intended to protect original expressions and not underlying functionality or ideas, the ability of courts to find an appropriate balance has been limited. Copyright doctrines including merger and *scènes à faire* should provide some assistance, but have proved difficult to apply in practice. Further, there are worrying judicial splits about when and whether the doctrines contribute to decisions about copyrightability per se as opposed to operating as a defense in an infringement proceeding to explain a substantial similarity between two computer programs.[234]

At the time, when Professor Ginsburg noted in 1994 that it was too early to retreat from copyright protection for software,[235] she was probably correct. In 1994, it was not clear what kind of a role patent, trade secrecy, and contract law would ultimately play in protecting code. It was also not clear how sophisticated DRM measures would become, and how quickly they might be developed to protect the interests of software producers. The ensuing decade has shown much activity in all of these areas. These developments, coupled with recent developments in programming methodology and digital copyright law (including the DMCA), suggest that large-scale copyright protection of code is simply not necessary anymore.

Software developers can now utilize the patent system to protect truly new and non-obvious software-related inventions and, to the extent that patents are not available, they can resort to trade secrecy, coupled with more effective contractual and DRM measures than existed previously. The "trade-secret-plus-contract-plus-DRM" approach can easily supersede copyright with respect to software because it addresses all the ills that copyright law was originally adopted to combat. Further, it combats those ills more effectively than copyright law because it focuses on the protection of functional ideas rather than original expressions.

That said, the question arises as to whether there is any harm in retaining copyright protection as a low-cost alternative for software developers to protect themselves against digital piracy. There is no specific *need* for copyright protection in light of the ample protections that can be provided through other legal means. In fact, copyright *can* potentially be harmful in the software code context in a number of ways. First, it is practically and theoretically unsatisfying to stretch copyright principles into areas for which they are not a good fit. Setting precedents

234. Lexmark Int'l, Inc. v. Static Control Components, Inc., 387 F.3d 522, 557, 559 (6th Cir. 2004) (Feikens, J., dissenting in part) (noting an existing circuit split on the application of the *scènes à faire* doctrine and that it is unclear whether the merger doctrine operates as a bar to copyrightability or rather as a defense to particular types of infringement).

235. Ginsburg, *supra* note 15, at 2560.

of this kind in the software code context potentially opens the door to further attempts to stretch copyright law into other contexts where it is not well-suited.[236]

Secondly, developments in programming methodology and digital copyright law are not static over time. As these areas develop, copyright appears to be a less and less comfortable fit for software code. Obvious examples of this are: (a) the increasing modularization and object-orientation of code which relies significantly on copying for innovation; and (b) the enactment of the DMCA, which is potentially causing problems of application where software code is incidentally incorporated into physical after-market replacement parts. Although recent judicial interpretations of both general copyright law and the DMCA's anti-circumvention provisions seem now to be limiting the reach of the legislation in these contexts, the door is still disturbingly open for inappropriate uses of this legislation in the future, as acknowledged by Judge Merritt in *Lexmark*.[237]

Further, as software producers increasingly rely on contractual, DRM, and patent measures to protect their code, the maintenance of copyright protection muddies the waters when disputes arise. Instead of courts being able to focus squarely on the valuable utilitarian elements of code, they are confronted with issues relating to the originality of expressive elements of code. These are difficult questions that add unnecessary time and cost to legal proceedings. This decreases the efficiency of the legal process in this area, and potentially disadvantages litigants who may not have the financial means to produce expert witnesses to support their claims and defenses. It also detracts focus from what is really at issue in the relevant cases, particularly as copyright protection does not appear to add anything very significant to the protections otherwise available to software developers through patent, trade secrecy, contractual licensing, and DRM measures.

In terms of the globalization concerns raised by Professor Ginsburg

236. One example of this is the debate about sui generis intellectual property protection for non-original database contents, such as a white pages telephone book. *See* Feist Publ'ns, Inc. v. Rural Tel. Serv. Co., 499 U.S. 340, 364 (1991). These would generally not be protected under copyright law in the United States on the grounds that they do not have sufficient originality to obtain copyright protection even under the low standards of originality required by copyright law. *See* LEAFFER, *supra* note 4, § 2.12. There have been some legislative moves, particularly in the European Union, to enact legislation based on a copyright model that protects such databases because of their clear commercial value. Council Directive 96/9, 1996 O.J. (L 77) 20 (EC). Such legislative models are arguably overly protective as attempts to apply copyright-like structures to non-copyrightable works. The Database Directive in the European Union has been criticized on this score: Jacqueline Lipton, *Balancing Private Rights and Public Policies: Reconceptualizing Property in Databases*, 18 BERKELEY TECH. L.J. 773, 777 (2003). It is, in fact, currently under review because of its potential over-protection of non-original works. Commission of the European Communities, *First Evaluation of Directive 96/9/EC on the Legal Protection of Databases*, COM (2005) (Dec. 12, 2005).

237. 387 F.3d at 551 (Merritt, J., concurring).

in 1994,[238] while it is true that most jurisdictions are harmonized in their adoption of copyright law to protect software code as a literary work, this approach does not have to remain in place if copyright law is causing problems of over-protection and innovation stifling. If the United States were to suggest scaling back copyright law in a particular context today, there may be less international consternation than there would have been in the 1980s or 1990s when copyright law was first adopted for code and had not been significantly tested yet. Now there is sufficient evidence to suggest that copyright law does not work particularly well for code.

Moving away from copyright protection of code would have to be carefully planned and executed to ensure that it is only software *code* removed from the ambit of copyright protection. It is appropriate to retain copyright protection, and DMCA protection, for literary, graphical, and other works reduced to digital formats, such as digital movies, music, eBooks, and even new forms such as video games and perhaps even some original and creative GUIs. Any item whose value truly lies in its actual appearance or expression as opposed to purely in its utility should still be subject to the protection of copyright law, even if the protection is only thin.

To remove copyright protection for software code, legislative action would be needed in both the United States and other jurisdictions. There would also need to be movement at the international level to remove the copyright protections for code in a number of relevant international treaties and European Union Directives.[239] Thus, the decision to eliminate copyright protection for software code is not a decision that should be taken lightly because it certainly involves much legislative and executive work both at the domestic and international levels. However, if something is not working well and is potentially impeding innovations in a particular field, it should be investigated with a view toward revising or eliminating it. A comment made by Professor Samuelson in 1994 seems even more pertinent today than it was at the beginning of the personal computer revolution:

> The status quo may seem comfortable, but it is unstable. Much of what is valuable in software cannot be appropriately protected by existing

238. Ginsburg, *supra* note 15, at 2562–63.

239. WCT, *supra* note 89 ("Computer programs are protected as literary works within the meaning of Article 2 of the Berne Convention. Such protection applies to computer programs, whatever may be the mode or form of their expression."); Agreement on Trade-Related Aspects of Intellectual Property Rights art. 10(1), Apr. 15, 1994, 1869 U.N.T.S. 299, 33 I.L.M. 1197 ("Computer programs, whether in source or object code, shall be protected as literary works under the Berne Convention (1971)."); Council Directive 91/250, art. 1(1), 1991 O.J. (L 122) 42 (EEC) ("In accordance with the provisions of this Directive, Member States shall protect computer programs, by copyright, as literary works within the meaning of the Berne Convention for the Protection of Literary and Artistic Works. For the purposes of this Directive, the term "computer programs" shall include their preparatory design material.").

> legal regimes, yet this very fact is what drives cycles of under- and overprotection. Left unaddressed, these cycles will not simply die down. Indeed, they are likely to grow worse as the industry matures.[240]

Although Professor Samuelson was arguing for the development of new sui generis forms of legal protection for code, today her concerns could be addressed by simply scaling back some of the current legal protections and relying instead on today's advanced technological protection measures supported by contract and trade secret law.

240. Samuelson et al., *supra* note 13, at 2364–65.