1-1-2000

The Paradoxes of Free Software

Stephen M. McJohn
Suffolk University, smcjohn@suffolk.edu

Recommended Citation
9 GEO. MASON L. REV. 25 (2000)

9 George Mason Law Review 25 Fall, 2000

The Paradoxes Of Free Software

Stephen M. McJohn *

* Professor of Law, Suffolk University Law School. Thanks to Lorie Graham, Jude McJohn, Bill McJohn, and Alex Behrakis.

Introduction

The open source software movement poses a profound challenge to the way that software is made and distributed. [1] Some of the best known pieces of software are open source: Linux, which runs many Internet servers and is the likeliest competitor to the scaled-up version of Windows; [2] Netscape Navigator, the browser that popularized the World Wide Web; [3] Apache, a widely used web server program; Sendmail, a common email server program; and the Perl programming language. [4] Open source software (also known, with somewhat different connotations, as free software [5] or open code [6]) differs in two key respects from most proprietary software. First, the holder of a copy of some open source software is free to make as many copies as she pleases, to modify the code, and to further distribute copies. [7] Second, to enable the foregoing, open source software is distributed with access to the source code, [8] not just the executable code version. [9] This article identifies the questions that open source software poses for intellectual property theory and doctrine, and concludes that open source software may have a considerable influence on the law of developing technologies--perhaps a greater effect than the law will have on software practices.

Most computer programs are written in source code, but run as executable code (also known as object code or machine code or binary code). [10] For example, here is a snippet of the source code from a program that runs on a gargantuan data storage device:

```
#include <time.h>

#include <limits.h>

#ifndef ACOS4

#include <sys.types.h>

#endif
```

Source code for a computer program is written in a high-level language, such as C or FORTRAN. Changes to the program are made to the source code, but the program is not run in its source code version. Rather, the source code is compiled (a process somewhat akin to translation or conversion) into executable code (a low-level language that uses the instruction set of the computer). Because the source code is more abstract, a few lines of source code can produce many lines of executable code. By analogy, if one could write a line of source code that said "Walk to the store," the resulting executable code might say, "Lift up your left foot. Stop lifting it. Move it forward thirty inches. Set it down." and so on until excruciatingly explicit directions for walking to the store were produced. Thus, a portion of the executable code produced by compiling the five lines of source code above would look like this:

```
F0F0F3F2 F0F0F0F0 F0F0F0F0 7B899583 93A48485F0F0F3F3 F0F0F0F0
F0F0F0F0 7B899583 93A48485F0F0F3F4 F0F0F0F0 F0F0F0F0 7B898695
84858640

F0F0F3F5 F0F0F0F0 F0F0F0F0 7B899583 93A48485

F0F0F3F6 F0F0F0F0 F0F0F0F0 7B859584 89864040

F0F0F3F7 F0F0F0F0 F0F0F0F0 40404040 40404040

F0F0F3F8 F0F0F0F0 F0F0F0F0 A3A89785 84858640

F0F0F3F9 F0F0F0F0 F0F0F0F0 A3A89785 84858640

F0F0F4F0 F0F0F0F0 F0F0F0F0 A3A89785 84858640

F0F0F4F1 F0F0F0F0 F0F0F0F0 40404040 40404040

F0F0F4F2 F0F0F0F0 F0F0F0F0 615C40E2 E8D4C1D7

F0F0F4F3 F0F0F0F0 F0F0F0F0 A3A89785 84858640

F0F0F4F4 F0F0F0F0 F0F0F0F0 40404040 40404040

F0F0F4F5 F0F0F0F0 F0F0F0F0 7B898640 84858689

F0F0F4F6 F0F0F0F0 F0F0F0F0 A3A89785 84858640

F0F0F4F7 F0F0F0F0 F0F0F0F0 7B8593A2 85404040

F0F0F4F8 F0F0F0F0 F0F0F0F0 A3A89785 84858640

F0F0F4F9 F0F0F0F0 F0F0F0F0 7B859584 89864040

F0F0F5F0 F0F0F0F0 F0F0F0F0 40404040 40404040

F0F0F5F1 F0F0F0F0 F0F0F0F0 7B898640 84858689

F0F0F5F2 F0F0F0F0 F0F0F0F0 A3A89785 84858640
```

```
F0F0F5F3 F0F0F0F0 F0F0F0F0 7B859389 86408485

F0F0F5F4 F0F0F0F0 F0F0F0F0 A3A89785 84858640

F0F0F5F5 F0F0F0F0 F0F0F0F0 7B859584 89864040

F0F0F5F6 F0F0F0F0 F0F0F0F0 40404040 40404040

F0F0F5F7 F0F0F0F0 F0F0F0F0 615C4040 40404040

F0F0F5F8 F0F0F0F0 F0F0F0F0 5C5C40E2 E8D4C1D7
```

Note that the foregoing executable code is even in a reader friendly hexadecimal format. As the computer would see it, it would simply be a binary wall of 0's and 1's four times as long. Like source code, executable code consists of instructions and data, but obviously in a much less user friendly form. Source code looks like stunted English (include this, do this, if X is true then add A to B), whereas executable code is an inscrutable mass.

When a person gets a copy of the program to use on her computer by downloading it from a web site, loading it from a CD, or otherwise getting a copy from the software provider, she usually gets just the executable code version. She can then run the program, but little else. If she would like to change the program, integrate it with other software, or analyze it to see how it works, the executable code version is usually not very helpful. Although the painstaking process of reverse-engineering may disclose how the program works, modifying the executable code would be much more difficult than simply modifying the source code and recompiling the program. [11] For someone who simply wants to run the program, the source code is unnecessary. However, for someone who wants to do anything else, the source code is generally required. Engineers often compare having the source code to having the ability to open the hood of a car, see how the engine works, and work on it.

For many software producers, the fact that their customer receives only the executable code is important. The producer attempts to maintain control over the code in two ways. She can deliver only the executable code, so the licensee can run the program but little else. [12] She can also deliver the code subject to a license that restricts further copying and distribution, so the licensee does not turn around and sell or give copies to other potential customers. Open source software producers, by contrast, grant much freer access, both practically and legally, by delivering source code along with the executable code, and by freely granting permission to modify and further distribute the software. As discussed in the following sections, open source software both challenges the theoretical underpinnings of intellectual property law and promises to affect the development of intellectual property law.

I. The Legal Structure of Open Source Software

Open source software is not in the public domain. Rather, a combination of copyright law and trademark law serves to permit the free distribution of open source software. The software is kept under copyright, but freely licensed under one of various open source licenses. The certification mark, "OSI Certified," may be affixed to a copy of the software to quickly show that it is open source. Anyone who takes a copy of the software can use it, change it, make and distribute more copies, and even sell copies without paying royalties to the original author. The open source license requires little; nevertheless, it does not abandon the copyright.

For example, suppose Ariel writes an original computer program, "Translator," which translates text from Swedish to Finnish. She could write the program in a number of ways: typing code at the keyboard, using a development tool program [13] that would produce detailed code from her directions, or speaking into a dictaphone and having someone else type it in, etc. [14] One way or the other, she produces the source code of the pro gram, but she might not be able to patent it. [15] Other programs exist that already do the same sort of thing, so the program (or more accurately, a process or machine embodied by the program) would be patentable only if it worked in a new way that was not obvious to people working in the field. [16] Even then, she would have to file a patent application, wherein she would describe the invention and claim quite specifically what was new about her process. [17] To date, however, open source developers have generally not sought patent protection for their software. [18]

Copyright, however, is much easier. As soon as the program is fixed in any form, the author has the copyright in the work. [19] The work needs only meet a very low standard of originality--quite literally, it only needs to be more original than a phone book. [20] Moreover, neither copyright registration nor any other formality is necessary to have the copyright, although registration is quite simple and helpful if she subsequently sues someone for infringement. [21] So, as soon as Ariel writes the program down on paper, on disk, or on tape recorder, she has the copyright. As a copyright holder, Ariel has a set of exclusive rights: the right to make more copies, to distribute copies to the public, or to make derivative works (i.e. to recast or transform the work into another creative work). [22] Anyone else who does any of these things would infringe her copyright, unless they have a license from her or unless they qualify for fair use or some other protective provision of copyright law. [23] She also has the exclusive right to display the work publicly or to perform it publicly, but since public readings or exhibitions of computer code are not too popular at the moment, [24] we will focus on the rights to copy, distribute, and transform the code.

Notably, Ariel does not need to publish her source code to receive protection under the intellectual property laws. She can register her program for copyright without disclosing much of the source code or executable code; rather, Copyright Office regulations require her only to disclose a portion of the code. [25] From that portion she may even redact any trade secrets or other proprietary material. [26] On the other hand, in order to obtain a patent, she must disclose the invention; however, such disclosure would only require a description of the invention used in the software that would enable another person working in the field to make and use the invention. [27] It would not require her to disclose the specific code she used to implement it, or the other code that comprised the rest of the program. [28] Thus, Ariel can receive a copyright with essentially no disclosure, and a patent with only a narrow disclosure. [29] Moreover, if she uses trade secret law to protect the program, publication is counterproductive. [30]

After writing the program, Ariel does not have a program that runs on a computer, rather just the source code text. Although she can store that text in a computer, she cannot run the program itself. To get the program into executable form, Ariel must run it through a compiler--a program that converts the source code into executable code. The computer can then execute the compiled program. Ariel then has the copyright in both the source code version of the program and the executable code version. [31] If Ariel wanted to change the program (e.g., to fix bugs or to add capabilities), she could change the executable code directly, which is an extremely laborious process, or she could change the source code and compile the program again.

Now suppose Beta would like a copy of Ariel's program--to use it, distribute it, and adapt it. Beta needs two things: a copy of the program, and Ariel's permission to make copies (or distribute copies, or to adapt the program). If Ariel were a typical commercial software company, Ariel would, for a sum of money, provide a copy of the executable code to Beta and license Beta to use the program subject to a number of conditions. The terms of the license would typically prohibit Beta from making additional copies, from distributing any copies (including the one that Ariel gave Beta), from reverse-engineering the program to figure out how it works, and from adapting the program--in other words, from doing anything other than using the program herself. Whether or not all those provisions are enforceable is unsettled, [32] but Ariel would likely take the position that the provisions are enforceable. Thus, Beta could face potential litigation from Ariel if Beta sold her copy to someone else, gave copies away to others, or made changes to the program itself. Moreover, if Beta wanted to change the program, she would have a hard time because she would possess only the executable code, not the source code.

Ariel could decide, instead, to distribute the code under an open source license. There are many reasons why she might want to do this. The leading rationales for open source are quite different. Some, most notably the Free Software Foundation, see it as an issue of ethics and politics. [33] Under this view, software is a form of expression. Imposing restrictions on this expression is as wrong as restricting the flow of scientific or artistic discussion. [34] Others see open source simply as a better way to develop software. [35] Some even call it a better business model, reflecting a far different world view. [36] If software is closed, then only the proprietor can change the source code. If the software is open source, then other developers are able to find problems or suggest improvements quite easily, leading to better software. [37]

To make the software open source, Ariel would provide Beta a copy of both the executable code and the source code. She could either give the code for nothing or charge for a copy. Either way, she would not give Beta the code free of restrictions; rather, Ariel would give it to Beta subject to an open source license. An open source license serves to keep the software free of any restrictions on use, copying, or distribution. As a pioneer of the open source movement put it, "Think free speech, not free beer." [38] Although Ariel might charge Beta for a copy, she would deliver that copy under a license that permitted Beta to do almost anything she wanted with the code.

There are various versions of open source licenses that Ariel could use. [39] Some open source licenses (such as the BSD license, the MIT license, and the Mozilla license, under which Netscape makes the code to its browser freely available) provide, in effect, that the licensee can do whatever she wishes with the software. [40] Other open source licenses require that code be kept open source. [41] Thus, if Beta does change the program and distributes copies of the new version, she must make both the executable code and source code available. Such an open source license prevents Beta from restricting the code legally, through licensing restrictions, and practically, by keeping revised versions of the source code under her control. Such licenses include the GNU General Public License (GPL) and the Artistic License.

The various licenses that grant access to source code all differ in some details. So what is a true open source license? The Open Source Initiative (OSI) has offered an answer in its Open Source Definition. [42] To be an open source license under that definition, a license:

1. must provide both executable and source code;

2. must allow modification and redistribution (with or without modifications);

3. must not limit distribution to certain fields of endeavor or products, or limit its use with other free software. [43]

To read a software license and determine whether it complies with those requirements is no easy task, particularly for a lay engineer who would rather read code than legalese. The OSI has provided an easy way for software developers to figure out if a license meets its definition of "open source" through the registration of a certification mark "OSI Certified." [44] Anyone who distributes software marked "OSI Certified" represents that the software is being distributed under a license that has been approved as conforming to the Open Source Definition. Therefore, through an elegant combination of copyright and trademark law, software can be easily maintained as open source.

An additional provision contained in most open source licenses is a complete disclaimer of warranty and limitation of remedies. For example, the GPL provides:

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL,

INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. [45]

In effect, an open source licensor quite reasonably says "I am providing you the source code. You can decide whether this software does what you need it to do." Moreover, most open source is distributed free of charge (as well as free of restrictions), so notions of risk-spreading by placing the cost on the maker are inapplicable. Returning to the hypothetical, if Beta wished to get a warranty from Ariel, presumably the two could negotiate such terms.

Open source licenses also address another trademark-related issue--protecting the reputation of the author of the software. [46] Even the most permissive open source licenses provide that if the licensee distributes the software, she must include the copyright notice giving credit for authorship to the original author. [47] Most open source licenses also provide that if the licensee modifies the software, then she must ensure that the modifications are not attributed to the original author. [48] She can do so by listing what changes were made, who made such changes, and when they were made. Thus, open source licenses require licensees to respect the author's right of attribution (to get credit for her work) and her right to avoid misattribution (not to have other people's work ascribed to her).

These attributes of open source licensing rest primarily upon the licensing protections of copyright and trademark. Most open source licenses do not specifically address the issue of patents, but open source developers are affected by patents as possible inventors or infringers. Suppose Ariel's program did contain a new, inventive process and she proceeded to patent it. In deciding whether to go open source or not, she could do so for any number of reasons. She might want to make money by licensing the patent, or she might want to patent the process in order to make sure no one else did, keeping the process free for public use. The patent gives her the exclusive right to use her process, or to make or sell a machine containing her invention. [49] By distributing her software under an open source license, she would authorize others to use the program free of her patent claim. [50]

The flip side is the risk of infringing someone else's patent. Even if Ariel devises her program independently, its use could infringe a patent she does not even know exists. [51] If Beta gets a copy and uses it, then Beta could likewise be liable. Beta might then consider suing Ariel for infringement of the warranty of good title, especially if Beta had paid for her copy. [52] The blanket disclaimer of warranty, similar to that of the GPL, may protect Ariel, as could a number of other arguments depending on the circumstances--such as whether Ariel charged for the copy, whether she was a software merchant, or whether the program was consumer software. [53]

The legal structure of open source, accordingly, is an elegant and robust use of intellectual property law. The net result turns the customary use of intellectual property on its head, by using intellectual property laws, which normally are used to guard exclusive rights, to safeguard free access to and use of software.

II. Open Source and Intellectual Property Theory: What Do Authors Want?

Various theoretical frameworks for intellectual property law have been brought forward in recent years. [54] At the risk of oversimplifying, one can lump them into three categories: economic theories, Lockean natural rights theories, and Hegelean personality theories. This part of the article first explores the economic theories of intellectual property and then considers the Lockean and Hegelean theories together as non-economic alternatives.

A. Law, Economics, and the Incentives to Give Away Work Product

The great difference in economic analysis of proprietary software development and open source development is well illustrated by the topic of network economics and industry standards. [55] For example, Mark Lemley and David McGowan discuss why for-profit software companies might not have the incentive to develop a potentially huge-selling product: the Windows operating system is protected by copyright law. [56] Copyright law, however, protects only the expressive aspect of works, not their functional aspects. [57] So another software company could copy the unprotected functional aspects of Windows and sell the functionally equivalent program to a big market without violating the copyright. But, as Lemley and McGowan explain, several reasons militate against such a strategy. [58] Although copyright does not protect functional aspects of the program, there remains legal uncertainty about which aspects are functional. [59] Also, other intellectual property (such as patents or trade secrets) might protect some aspects of the program. [60] More important than the legal uncertainties, perhaps, the market risks would be great deterrents to a commercial competitor. Reverse-engineering the program is a time-consuming and uncertain enterprise, and Microsoft periodically upgrades the program, which means that a commercial competitor might have difficulty in selling an up-to-date product. [61] Consumers might also be wary of whether the program was truly a functional substitute. [62] Finally, and most dulling to the incentives, Microsoft presumably has the ability to lower the price of its program to compete with any new entrant, so the potential payoff is greatly reduced. [63] All in all, it makes little sense [64] for a commercial competitor to make the huge investments in development and marketing that would be required to compete--when other avenues of investment are likely to be more fruitful.

The economic incentives for open source developers can be quite different. Certainly, some open source developers simply wish to sell soft ware, and would thus be subject to the same disincentives. [65] For many open source developers, however, the incentives are quite different: enjoyment of programming itself, the desire to show off technical feats to others in the field, the wish to sell software-related services, an idealistic urge to further computer science, and even the desire to tweak the proprietary software companies. Nor would they be scared off by the fact that an existing software product seller could respond to a new rival by lowering prices--because the open source developers are giving their product away anyway. Therefore, open source developers might be willing to take on a task, such as building a Windows emulator, where a profit-seeking enterprise would not. Indeed, just as economic theory might predict, such an enterprise exists: the WINE project [66] is an open-source project building a Windows emulator to run with the Linux operating system--in short, a piece of software that copies the functionality of Windows running on a Linux system.

So the economics of open source may be somewhat different from the economics of more established areas of intellectual property. Intellectual property laws, however, provide some of the incentives fueling the current open source movement. Remember that open source software is not public domain software: the developer does not renounce her copyright and put the code into the public domain. Rather, the code is freely distributed, subject to one of the open source licenses. So the question becomes, what does open source teach about the underlying economic theory of intellectual property? Economic theories of intellectual property law have obvious appeal, both because of the increasing role of intellectual property in economic life and because the United States Constitution seems to have an explicit economic basis for intellectual property law. [67] These theories come in several flavors, with the common theme that intellectual property laws should promote economic efficiency. These theories include the incentive, property, and spartan theories.

1. The Incentive Theory

The incentive theory draws its legal basis from the Constitution: "The Congress shall have Power . . . to Promote the Progress of Science and the useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries." [68] Under this approach, intellectual property law exists in order to overcome the free-rider problem with public goods. [69] Without copyright law, the thinking runs, authors have a diminished incentive to produce books, songs, and computer programs. To give authors an incentive to write, copyright law gives them the exclusive rights to make copies of their books and to sell these books. Likewise, the award of a patent grant gives a pharmaceutical company an incentive to spend millions on research.

The flip side of the incentive approach is that intellectual property laws should be limited to the protection necessary to promote the creation of new works and inventions. Language of several Supreme Court cases appears generally to follow the incentive approach, characterizing copyright as an incentive to create works without burdening users of the work. [70] Nevertheless, the incentive theory clearly does not comport with the existing state of intellectual property law in the United States. Intellectual property rights have been steadily expanding for the last several decades. Copyright law presents perhaps the greatest disparity between theory and reality. Copyright now applies to all creative works and is no longer limited to published works where the author chose to reserve copyright explicitly. [71] The enforcement term has steadily expanded, and now stands at an improbably long life plus seventy years. [72] Special statutory protection has been given to anti-copying technology and even to the printed terms and conditions on copyrighted works. [73] Such measures take copyright far beyond the scope necessary for the incentive to create works and eclipse any notion of balance. Rather, as industry has increasingly guided the legislative agenda, the copyright statute has gone from a rather simple framework to a labyrinthine monster, full of lengthy specialized provisions. Other areas of intellectual property have likewise expanded.

The incentive theory, then, stands more as aspiration than a description of intellectual property law. One recent case tested whether the courts will strictly limit Congress' power to pass intellectual property laws on the incentive rationale. In Eldred v. Reno, [74] plaintiffs sought to have the most recent extension of the copyright term declared unconstitutional. The Sonny Bono Copyright Term Extension Act [75] increased the term from life plus fifty to life plus seventy, and increased the term for work-for-hire from seventy-five to ninety-five years. Most importantly, the Act was retroactive, so works created in the 1920's (such as early Mickey Mouse cartoons, in his Steamboat Willie incarnation) that were about to enter the public domain will now remain under copyright. The plaintiffs in Eldred challenged the retroactive extension on the logical ground that an extension of the copyright term by Congress in 1999 is hardly going to affect the incentives of authors working in 1920. Nevertheless, given the Supreme Court's generally broad reading of Congress' Article I powers (such as the extremely broad reading of Congress' power to regulate interstate commerce, which the Court has read to permit Congress to legislate on practically any commercial matter [76]), the Eldred plaintiffs faced a steep uphill battle. [77] The District Court for the District of Columbia ultimately decided that the extension of the copyright term was "within the discretion of Congress." [78] Meanwhile, the incentive theory is an attractive theoretical rationale for intellectual property laws, but it does not comport with the increasingly broad laws in the United States and abroad as under such treaties as TRIPS. [79]

2. The Property Approach

The property approach, by contrast, favors extending intellectual property protection much further than the incentive approach. [80] As with the incentive approach, the ultimate goal is efficient use of resources. However, the property approach looks not only to providing an incentive to create works, but also to providing an incentive to exploit those works efficiently. The strongest economic justification for private property generally is that it internalizes costs and benefits. Under the classic "tragedy of the commons," if property is held communally, then perverse incentives are created. An individual might use up resources inefficiently because she does not bear the cost or might fail to create productive works because she does not reap the rewards. The property approach has been roundly criticized for, among other things, failing to give sufficient value to the public domain. [81]

3. The Spartan Theory

The spartan theory is a reaction to the great recent expansion of intellectual property law. [82] Under this approach, intellectual property rights are seen as a necessary evil. Copyright restricts freedom of expression; patent restricts research and the utilization of technology; trademark restricts competition. All three types of restrictions have countervailing benefits, such as overcoming the free-rider risks inherent in some types of information-producing endeavors. However, due to the high costs such restrictions impose, the intellectual property laws should be sharply tailored [83] in a way reminiscent of the First Amendment, which only tolerates restrictions on speech that are sufficiently narrow. For example, copyright should not attach broadly, as it does now, to all areas of human creativity. [84] Rather, copyright should only apply to works to the extent that copyright protection is necessary and desirable to promote the creation of such works. [85] Likewise, trademark protection should be more strictly limited to prevent consumer deception, in contrast to the broad protection of brand names under existing law. [86]

4. Can Open Source Fit Within an Economic Theory of Intellectual Property Law?

Open source can be seen as inconsistent with all three economic approaches. If an economic reward is necessary to spur creation of works, then there would not be thousands of developers writing software only to freely give it away under an open source license. Thus, the incentive theory of copyright seems to be inapplicable. Likewise, success of the free distribution and alteration of open source software seems to refute the property approach. Rather than the creator of software (or useful additions) reaping the rewards, it is freely distributed. Even the spartan approach looks positively generous, granting a minimal level of copyright protection, where maybe none is necessary. Thus, open source software might be seen as confirmation of the view that the regime of intellectual property law has been outmoded by digital technologies; however, closer examination shows that economic reasoning underlies much of the open source framework, and that open source has anticipated recent developments in economic legal theory.

The spartan approach most clearly resembles the spirit of the open source movement. The open source movement is based on the idea that software should be freely distributed and revised, and in the eyes of some, that restricting access to source code is morally wrong. So if there were to be intellectual property in software, one might think it should be as minimal as possible. However, as discussed above, open source developers rely on intellectual property laws to prevent certain uses of open source soft ware, such as its incorporation into proprietary products or its distribution without proper attribution of authorship. [87] Thus, adopting a minimalist approach to intellectual property law could harm the open source movement, if it reduced the ability to further the movement's goals.

For the same reasons, open source is reconcilable with the incentive theory of intellectual property. Many open source authors are spurred to create code by incentives other than copyright: the love of elegant problem solving (a.k.a. hacking), status among their peers, the wish to further computer science and make things better generally, and even animosity toward commercial software developers. An important factor in whether a programmer is willing to share her code is whether others might try to free-ride on her efforts to make a profit. Such a result agrees with more recent trends in law and economics. Richard McAdams has shown how law and economics has paid far too little attention to the effects of status considerations on incentives. [88] People are motivated not just by purely material considerations, but also by their desire to achieve status in the eyes of others. [89] Accordingly, economic analysis of law must not only consider how legal rules affect wealth in terms of possessions, but also in terms of the estimation of one's peers. [90] Otherwise, the true incentive effects of laws will be obscured. Open source provides a particularly striking example. Looking only to material considerations, open source developers might appear to be acting contrary to rational economic incentives, by giving away software. However, when one considers the return in terms of increased status among software developing peers (i.e., showing off technical prowess, or receiving approval for participating in the open source movement, or building relationships in the development process), ample incentives become apparent. Likewise, open source licenses that require the code to remain open source can protect against a possible disincentive--the resentment that a developer might feel if she gave away software only to see it incorporated into a proprietary product.

The strangest intersection of open source and legal theory, however, is how well the open source movement can be used as an example in favor of the property approach--the theory that intellectual property rights should be as expansive as practicable. The underlying idea behind the property approach is not that authors and inventors should be enriched. Instead, the property approach takes a privatization rationale: intellectual property laws serve to promote the more effective use of information, by giving individuals the incentive to exploit it, rather than letting free access by all lead to waste. Recall that the open source movement relies heavily on keeping software under copyright and distributing it under a license, rather than putting the software into the public domain where anyone can do with it what she will. The open source licenses impose two key restrictions (or, more accurately, restrictions on restrictions): (1) the licensee may not restrict distribution of the code and (2) the licensee must make the source code available to others. In addition, most open source licenses add other restrictions, such as the requirement that authorship of the code be properly attributed. In a world without copyright, such licenses would obviously be ineffective; however, even in a world with relaxed copyrights, such provisions would likely be ineffective. [91] Only under a legal regime like the present, where copyright holders have great control over their works, and licensing revenues are not at issue, are such restrictive licenses likely to be enforceable.

Accordingly, the open source movement stands as a counterexample to one of the strongest criticisms of the property approach--the transaction cost argument. This argument runs as follows: although privatizing intellectual property could, in theory, lead to efficient exploitation, transaction costs prove a formidable obstacle. There are many uses of intellectual property that would never occur, due to the inability of the rights holder and the potential user to reach an agreement for a license. For example, if a book is under copyright, a teacher might wish to make a copy of one chapter to hand out to her class. It may well be that the copyright holder would not object, but the transaction costs necessary would prevent the agreement from occurring (e.g., the teacher must identify the rights holder, communicate with her, negotiate and execute a license). To the extent that transaction costs prevent use of intellectual property, while at the same time not enriching the holder because no agreement is reached, there is a "dead-weight loss" of the kind so abhorred by economists. Thus, one strong argument against expansion of intellectual property rights is that exceptions are necessary to prevent waste, such as applying the fair use doctrine in the teacher example. But the response of the property theorists is that market mechanisms will arise to overcome such transaction cost problems--for example, performing rights organizations like the American Society of Composers, Authors & Publishers (ASCAP) and Broadcast Music, Inc. (BMI) have made it possible for thousands of copyright music holders to negotiate licenses with millions of potential users. [92] Likewise, the open source licenses solve a similar collective action problem. By using open source licenses to coordinate the diverse group of open source developers, their common goals can be reached efficiently. Ironically, then, the open source movement, with its early roots in a decidedly socialist view of software, appears to vindicate a rather free-market view of intellectual property--that market mechanisms are more efficient in overcoming market failure than corrective legal measures.

B. Philosophy and Open Source

As discussed above, intellectual property law in the United States draws its justification from economics. Other countries look more toward philosophical underpinnings for intellectual property. [93] For example, the Universal Declaration of Human Rights provides that an individual enjoys the "right to the protection of the moral and material interests resulting from any scientific, literary or artistic production of which he is the author." [94] Philosophical underpinnings for intellectual property are conventionally divided into the natural rights theory associated with John Locke and the personality protection theory associated with Friedrich Hegel. [95]

The Lockean analysis is somewhat difficult to square with open source practice. As interpreted by modern scholars, Locke would consider intellectual property protection appropriate where "the production of ideas requires a person's labor; second, that these ideas are appropriated from a 'common' which is not significantly devalued by the idea's removal; and third, that ideas can be made property without breaching the non-waste condition (i.e., will not let the idea be unutilized, like fruit that perishes for lack of use)." [96] Open source software fits strangely into that theoretical framework. The first condition is easily met. Writing software certainly requires a person's intellectual labor. Even in these days of sophisticated development tools, software requires considerable work to design, implement, debug, and revise. The third condition (non-waste) would also seem to be met. Arguably, some open source licenses cause a kind of waste in that they prohibit incorporating open source software into proprietary products, and thus foreclose many uses of the software. Such gentle restrictions, however, leave open many other uses of the software. Moreover, because copyright law prohibits only near verbatim copying of software, a proprietary producer is still free to copy the functional aspects of the open source software. Thus, the first and third conditions seem to be met.

The second condition, however, poses a conundrum. Under this central idea to Locke's justification of property, property can only be appropriated if the net effect does not diminish the commons. Thus, a tract of land can be put into the private hands of a farmer because she will then have an incentive to use it productively and sell her harvest to the public. Such use is more productive than leaving the land to lie fallow. One could argue that open source passes the condition by definition. The whole point of an open source license is to leave the code open for use by others. Thus, rather than diminishing the commons, the copyright in open source software protects the commons.

There is a distinction between the public domain and the common available to open source users. Where the open source license is conditioned on maintaining the open source nature of the software, the code is not in the public domain--rather, it is usable freely by those who agree to abide by its conditions. Those conditions, in turn, are rather generous--requiring only that users agree not to impose conditions of their own on the fruits of the code. Nevertheless, such conditions do decrease the stock of ideas available to all, but only to the extent of the rather thin copyright protection afforded software. So, attempting to fit open source into a Lockean framework eventually begs the question, or perhaps raises the more complicated question, of whether property is best owned individually or collectively.

Another theoretical basis for intellectual property protection is the idea that creative works and inventions embody the personality of the artist or inventor, and accordingly should be protected in the same way that tort laws protect people from physical and intangible harms. [97] In this view, legal protection of intellectual creations is necessary to permit individuals to achieve self-actualization. As a descriptive matter, open source seems to favor the personality approach over the natural law approach. One can view open source as an experiment to answer the question, "What do creators most care about?" As noted above, open source developers will produce code without the conventional economic incentives provided by copyright protection, and will produce code much more generously than one might predict with a Lockean framework. They will add to the commons of open source code without taking anything out, provided that others do the same. However, open source developers do not give up the protection of their reputation. As noted above, open source licenses freely give up almost all exclusive rights (e.g., allowing free copying, distribution, modification, and so on) except rights of attribution. [98] To the contrary, they strictly require that the original author receive credit for her work, and that any changes in the code are not misattributed to her. [99] So open source seems to show that perhaps the most important thing to many productive authors is their reputation, because the one thing they will not give away is their name.

Here, the legal rule chosen by open source developers reflects the European approach to intellectual property rather than the American. The protection of reputation--that the work be properly attributed to the author and not be misattributed to her--is something that is at the core of copyright in many European systems, under the rubric of "moral rights." [100] In the United States, by contrast, moral rights are much more limited. In order to become a party to the Berne Treaty, [101] the United States was obliged to provide some protection for moral rights. However, Congress chose nearly the minimal extent of protection that would pass muster; it added some protection for moral rights, but only to a narrow category of works like paintings and sculpture. [102] The open source movement shows, however, that creators of much more functional works, such as computer software, are no less interested in their status as authors and the fate of their works.

III. Patents: Reshaping the Prior Art Problem

The relationship between computers and patent law has altered radically over the few decades of electronic computing. The sad story of the patent litigation surrounding ENIAC, [103] the first general-purpose electronic computer, illustrates the disregard of legal niceties that computer pioneers once allowed themselves. John Mauchly, Presper Eckert and their crew (with assistance from the redoubtable John Von Neumann) spent many long years and overcame dozens of theoretical and engineering obstacles to make the ENIAC function. [104]

Only belatedly did they attempt to secure patent protection on one of the century's most influential inventions. Because they had waited so long, several daunting legal obstacles prevented them patenting the ENIAC. [105]

By contrast, one of the first concerns facing today's commercial software developers is whether they can patent their work. [106] In part, this simply reflects the much greater attention that scientists and engineers now pay to intellectual property law. To illustrate, the 1999 Nobel Prize winners in medicine and chemistry emphasized that their work decades earlier had been driven by a desire for knowledge, not profit, to explain why they had not patented their discoveries. [107] On the other hand, recently, researchers at MIT reported that they had inserted a gene into a transgenic mouse, thereby decreasing the reduction in memory with age. [108] Notable was how patents pervaded the story--the gene used was already covered by someone else's patent, and the researchers, long before they told the New York Times or the rest of the world about their work, had filed for another patent on this novel use of the gene. [109] Scientists and engineers are hardly intellectual naifs any more.

Another reason that computer developers, both in hardware and software, pay more attention to patents is that the law on patentability of computer-related inventions has itself changed radically--from forbidding to welcoming. Even when the ENIAC developers finally got around to seeking patents, they sought only to patent the hardware, not the software. Similarly, in succeeding decades, little attention was paid to whether there were intellectual property rights in software. Rather, the software was simply included for free with the hardware as part of the package. Early case law viewed attempts to patent software skeptically.

In a series of cases, the courts have moved from denial of patentability to an open door policy. [110] In 1972, the Supreme Court denied patentability to a process to convert binary numbers coded in decimal form to pure binary numbers. [111] The Court reasoned that the claimed invention was simply a way of solving a mathematical problem, and therefore unpatentable. [112] In 1978, the Court, for similar reasons, held unpatentable a process that used a mathematical formula to update an alarm limit, a number calculated to monitor operating conditions in a catalytic conversion process. [113] These cases cast great doubt on the patentability of software, because broadly stated, computer programs always take numbers as input and produce numbers as output. But the tide turned in 1981, when the Court held that the use of a computer program to monitor a rubber-curing process was a patentable invention even though the only novel aspect of the process involved calculating numbers. [114] The Supreme Court has not addressed the issue since then.

The authoritative Federal Circuit, however, after struggling to draw a line between nonpatentable mathematics and patentable useful processes, greatly relaxed the scrutiny in a series of cases. [115] Under existing law, it is likely that the processes rejected under the early Supreme Court cases would have been held patentable today. As a result, patents on software, especially software implementing business methods, have become increasingly important. [116] Now, software may be patented in many ways: as a process, a component of a machine, or as an article of manufacture. [117] Indeed, recent software patents extend to a propagated electric signal as an article of manufacture. [118]

Some think that software patents may pose the greatest threat to open software. [119] After considerable reduction in the legal obstacles to patenting software, many thousands of software patents have been issued in recent years. Open software developers might write code that allegedly infringes such patents. Richard Stallman, a leader in the free software movement, has likened software patents to a minefield for open source developers. [120]

As some have noted, an open source defendant does have one particular disadvantage, as compared to other software developers who might be potential patent infringers. [121] This risk arises from the very nature of open source software. Suppose someone holds a patent on a process used in software--a process for sorting data, or for producing a particular format of output. If a proprietary program used the patented process, the patent holder might not be able to find that out. The process might be used in the program, but not in a way that was evident to a user of the program. One could tell that the program was, at some point, sorting data, but would have to go to considerable trouble to figure out how the program was sorting it. Indeed, that would be impossible if one did not have access to a copy of the program. It would be much easier, in some respects, to monitor open source programs for infringement of the patent, for the very two reasons that make them open source--one would be entitled to get both a copy of the program and a copy of the source code. So in one respect, open source is peculiarly susceptible to patent monitoring.

Another area in which open source developers could be at a disadvantage is in cross-licensing. Because so many software patents have been issued in recent years, and perhaps because the validity and enforceability of many of the patents is rather unclear, patent licensing is quite different in the software area than in other high-tech areas such as biotech. In particular, royalty-free cross-licenses are quite common in the computer industry. The parties to such licenses agree, in effect, not to attempt to enforce their patents against each other. Such nonaggression pacts protect only the parties to the license. To the extent that open source developers do not seek software patents, it may leave them out of such protection, having nothing to offer as a quid pro quo.

Open source developers may have other advantages that more than make up for such potential risks. Indeed, the open source software movement may well redirect the course of software patent litigation in several ways. The greatest issue at present in software patent law is the problem of prior art. Patent law provides that an invention is only patentable if one concludes, after examining the prior art, that the invention is both novel (is not already known in the prior art) and nonobvious (would not be obvious to a skilled worker in the field, in light of the prior art). [122] What constitutes prior art is defined rather tortuously in the statute, [123] but one can think of the prior art as being the stuff in the public knowledge.

Computer software, however, is a difficult field in which to locate the prior art, for two reasons. [124] First, as discussed above, software has only gradually been seen as patentable, so there is not a great stock of software patents to provide a source of prior art. Second, the prior art in computer science is much less organized than in many other fields. In other new technologies, such as biotech, it may be relatively straightforward to check scientific journals and other sources to see if a claimed molecule is in fact novel. [125] Computer programming, by contrast, has had much less systematic archiving of knowledge. Much of the knowledge in the trade is in informal form. And more recently, much of the knowledge was intentionally kept out of the public domain. One commentator has determined that some eighty percent of issued software patents make no effective citations of prior art, despite the great amount of published work in computing. [126] Recognizing the special problem of prior art in the area of computer related inventions, the U.S. Patent & Trademark Office (USPTO) has begun a project to more systematically organize knowledge in the computer arts, while several private bodies offer help in locating prior art. [127] In addition, the fact that so many software patents have been issued in recent years will make a considerable contribution to the amount of prior art that is available for searching.

In the meantime, a defendant in a patent infringement action may have a very difficult time proving that a technique was already in the prior art, or was obvious given the prior art. An open source defendant, however, may have a card to play that is unavailable to other defendants. The recent activity around several controversial patents illustrates how an open source defendant could prove a veritable Hydra of a defendant. [128] Inventors had succeeded in obtaining patent protection on several widespread technologies: fundamental techniques of multimedia, [129] a commonly used hack ("windowing") to fix year 2000 (Y2K) problems in aging software programs, and a privacy protection algorithm that threatened to control a common Internet standard. [130] In each case, widespread publicity about the patent, together with considerable anger that someone claimed to have invented something that other programmers considered old hat, resulted in programmers sending many examples of patent-invalidating prior art to interested parties and the USPTO. In each case, the tide turned--the USPTO took the unusual step of initiating reexamination of the multimedia and Y2K patents, and the privacy patent likewise looked questionable. [131] Open source developers, such as the world-wide Linux network of thousands of software developers, present a formidable resource for locating prior art--and likewise have shown their willingness to spring into action in defense of the movement.

If the prior art shows that the invention is not novel, the patent can be invalidated. Even if the invention is novel, it is still invalid if it was obvious in light of the prior art. This is a particularly difficult determination with new technologies. [132] Here, open source also may benefit from its moral suasion and from the favorable opinions of its many experts. The case of the ENIAC throws some light on how the identities of the parties, and the likely effect on industry could influence courts sub silentio in software patent cases. The judge in the ENIAC case denied a patent to Eckert and Mauchly on the ground that they had derived the idea from John Atanosoff. [133] That did not mean, however, that Atanasoff could now patent the ENIAC, for the passage of time now acted as a statutory bar. One way to interpret the result is that the judge may have been attracted to a ruling that did not put the new technology within the exclusive control of one party. Likewise, open source developers could be very sympathetic parties, and courts may lean, given that the technology and the law is sufficiently complicated, toward restricting patent coverage--as opposed to cases where two parties are simply fighting about which one gets to keep the technology out of the public domain. This is hardly a cheery view of judicial decision making--ad hoc results-oriented rough justice in patent cases--but may prove a realistic one, and at the least a reason to avoid opening the Pandora's box of patent litigation. [134] Another possibility would be for open source developers to fight fire with fire, by seeking patents of their own and combining them. [135]

In addition to proving a difficult defendant to beat, some open source developers might be pointless to beat. Patent law remedies such as actual damages, punitive damages, and injunctions may be worth little against some open source defendants. Actual damages are most often based on recouping defendant's profits from the use of the patented invention and on compensating for plaintiff's losses. Both types of damages could be difficult to prove. At this point, the vast majority of open software developers do not charge for their software, so there would be no gains to recoup.

On the plaintiff's side, proving loss of revenue could be very difficult. Plaintiff, in effect, would have to prove that if defendant's free product had not been available, buyers in the market would have paid money to buy from plaintiff, or from people using plaintiff's invention under a license from plaintiff. Nevertheless, the giveaways during the browser wars have shown, and many a doomed business on the Internet has learned to its distress, free stuff gets a great many more users than stuff with a money price. Punitive damages would also be unlikely to make up for lack of actual damages, because an open source infringer would likely have acted innocently, unaware of the patent at issue. Nor would an injunction necessarily help, given the nature of software. In intellectual property cases, the ability to get an injunction is a huge weapon for plaintiffs. To be able to get not only damages, but to also stop a defendant from showing her movie, using her trademark, or employing a key hire, is often the greatest benefit of successful litigation. However, software is a little different. As software developers say, there is more than one way to skin a cat. If a program cannot use a particular patented algorithm, then most likely the developer can simply devise a different one that would have to fall outside the terms of the patent and the doctrine of equivalents. [136]

An additional reason why many open source developers are unlikely to be defendants is that they do not have the deep pockets that attract plaintiffs. One could see a software company bringing a case not to bring in funds, but rather pour encourager les autres. Open source developers are quite different, however, from defendants often sought in other areas of intellectual property enforcement to make a point about willingness to enforce intellectual property. A software or music publisher might go after pirates not just to seek damages, but also to set an example as a deterrent to others, and to encourage them to apply their skills in an area less aggressively enforced. Such actions against software developers could have quite negative effects--a hacker innocently reinventing an algorithm is a much more sympathetic figure than a bootlegger. Moreover, the publicity of the enforcement could backfire another way.

As noted, one response is to rewrite the software with noninfringing code. If the potential defendant does that, and is an open source developer, then that substitute for plaintiff's invention will now be freely available for others to use and advertised by the lawsuit itself. This result provides yet another reason to tread very quietly before such unexpectedly dangerous defendants.

IV. Trademarks: a Banner

Most trademarks are created with little attention to trademark law. An individual practicing her trade or a little business starting in a garage will likely use a name intending to do the sort of things trademarks do: to build a name among potential clients and partners, and to distinguish its goods or services from competitors. However, the business is likely to ignore the niceties of trademark law--what sort of symbols may be trademarked, what protection the trademark laws give, and how to register a trademark. Indeed, many start-ups are likely to confuse the various types of intellectual property protection and talk about copyrighting or patenting the name of the business.

The open source movement comprises software developers, not lawyers. Nevertheless, mindful of the way intellectual property laws had caused them problems in the past, open source developers showed considerable sophistication in their use of trademark law and, even where lack of legal experience caused a bump, recovered in a way that compares favorably to the legal strategies of the most sophisticated commercial actors. The typical start-up simply uses a name and then at some point thinks, "Hey, maybe we should register our name as a trademark," or more likely, proceeds in blissful ignorance until they have enough success (or trouble) that they have consulted a lawyer. Even the most sophisticated business actors can ignore trademark issues.

The open source movement, by comparison, made sophisticated use of trademark law. One approach would have simply been to register a trademark for the software products and distribute it under that name. Such an approach would have raised two big trademark law problems with the open source model of distribution. Open source software can be freely adapted and distributed further, by people other than the original producers. However, a trademark must identify the source of goods. [137] If adapted and redistributed software bore the original mark, it would be a misleading use--unless somehow everyone producing open source banded together as a single producer. In addition, the trademark holder must police the use of her mark. Just as a trademark holder cannot make a naked assignment (i.e., sell the mark for someone else to use), so she cannot simply allow others to use the mark without verifying that their goods or services conform to her standards. Thus, typical use of trademark would have led to problems down the road. The open source movement, however, turned to a refined use of trademark law--the "certification mark." Unlike most marks, the mark is used not by its owner, but rather by others to indicate that it meets standards set by the mark owner. Thus, the "Underwriter's Laboratories" (UL) mark is used by manufactures who comply with the relevant standards. [138] Likewise, the Open Source Initiative decided to register a mark that it would permit others to use if their software complied with the Open Source Definition.

The initial mark chosen, however, reflected a common trap for the unwary in trademark law. The first mark chosen was "Open Source," using a recently coined term that succinctly described the movement to make source code freely available. [139] The apt nature of the mark made it questionable as a matter of trademark law. As many a business does, the open source developers sought to find a name that was as descriptive as possible. They opted to register the mark "Open Source." [140] Before too long, however, it became clear that the USPTO would likely reject the mark on the grounds of descriptiveness. A mark cannot be registered if it is simply descriptive. That left several choices. One could appeal the decision of the USPTO. Certainly, various high tech companies have resisted USPTO determinations that marks they sought to protect were invalid as descriptive or generic. [141] One could also seek to register the mark on the supplemental register, then get a principal registration after it had sufficient publicity to acquire secondary meaning. [142] That may have ultimately been successful, given the great publicity given to various open source efforts and its widespread currency in the software field. The open source initiative, however, made what appears to be a much wiser choice. They simply changed marks, something businesses, especially once egos become involved, can be highly reluctant to do, and they registered the certification mark "OSI Certified." [143]

Notably, the open source initiative did not give up on their claim to the phrase "open source." Although they abandoned their attempt to trademark it, they have other resources to call upon. There are other ways to grant accessibility to the source code that would not qualify as "open source" under this definition. For example, Sun Microsystems recently published the source code of its Solaris operating system and permitted developers to make modifications to the code. So in a sense, the source code to Solaris is now open: it is open to public examination and can be modified. However, Sun's license contained restrictions that go beyond those permitted to qualify as "OSI Certified." So is Solaris open source? The important thing is that the open source initiative can look beyond the law for assistance in using its mark, in a way that most mark-holders cannot. Xerox, for example, would prefer that people refer to "photocopies" rather than "xeroxes," for the latter term leads down the road toward common usage; however, few people will feel a moral (or grammatical) obligation to use a trademark only in its trademark sense. Some software developers are already sensitive to uses of the term "open source" in what they consider to be inaccurate uses of the term. The reliance on community norms of language, as opposed to law, has not always worked for software. Most notably, the original meaning of "hacker"--one skilled in programming--has lost the linguistic battle of the standards, as the now popular understanding of "hacker" is a malevolent system breaker, or a "cracker" to the original "hackers." [144] However, with the more technical term "open source," the experts' opinion may more likely prevail.

V. Regulation

Whether, and how, to regulate the Internet has spawned huge amounts of discussion in past years. [145] In Code and Other Laws of Cyberspace, [146] Lawrence Lessig introduces two important elements to the discussion of regulation of computer networks such as the Internet. First, he makes clear why the commercial law governing software transactions could have momentous effects. As Lessig explains, computer code itself is a form of regulation, because it imposes constraints on the behavior of users. [147] Accordingly, the ability of software distributors to control what is done with their software has considerable effects beyond the immediate parties to a software transaction. Thus, the content of software licensing law may be more momentous than similar law that governs other commercial transactions. [148] Lessig also makes a rather far-sighted point about the ability of government to control software. He argues that open source code is more difficult to regulate than proprietary code, because of the diffuse nature of the open source model. [149]

A. Licensing Law and Open Source: Fair Use in Copyright and a Price Less Than Zero

Software licensing is a burgeoning area of law practice. [150] Billions of dollars are spent on software that is generally provided under a license. Strangely enough, relatively little is known about the legal effectiveness of software licenses, as compared to older segments of the economy. For example, Article 2 of the Uniform Commercial Code (UCC) provides a comprehensive set of laws governing contracts for the sale of goods. [151] How to form a contract, what terms automatically become part of the contract, whether the seller can disclaim warranties and limit remedies, and what the measures of damages are--all essential terms are governed by a uniform set of rules that has been adopted by all fifty states.

The state of software law, however, is far less settled. There is no definite authority on such basic matters as, whether a sale of software is covered by the UCC, whether the most common form of software licenses (shrinkwrap and clickwrap agreements) are legally binding, [152] whether warranties can be effectively disclaimed, or whether copyright law preempts provisions in software licenses. [153] The lack of legal clarity certainly has not prevented software from becoming a gigantic industry; however, various efforts have been made to make software law clearer and more uniform. The best known project was the drafting of a proposed addition to the UCC--draft Article 2B [154]--but the path to the state legislatures has not been smooth. Not surprisingly, meetings to discuss the draft UCC 2B were not attended by all possible parties, especially consumers. [155] Consumers often do not bother to read contracts at the time they enter into them. Thus, a consumer is highly unlikely to pay much attention to a lengthy drafting process, which might produce a law, which might govern a software contract the consumer signs years in the future. On the other hand, a software company has a direct interest in the law governing sales of its product. The draft 2B struck many commentators as being crafted more with the interests of sellers rather than buyers in mind. [156] Alternatively, a successor model statute, the Uniform Computer Information Transactions Act, [157] has been offered to the states. As commentators and lawmakers consider what rules should govern licensing, open source licensing presents the issues in a fresh form. This Part considers two key issues: the application of fair use to open source software, which would obviate the need for a license, and whether licensing law should be strengthened to bolster open source licenses.

The various open source licenses are some of the most generous intellectual property licenses around. A typical proprietary software license provides, in effect: "Here is the software you paid for. You can use it yourself, but you do not have permission to make any more copies, to change it, or even to analyze it in order to figure out how it works." By contrast, an open source license tells the licensee, in effect: "Here is some valuable copyrighted software, free of charge. You can make as many copies as you want, you can adapt it however you like, and you can give away or even sell copies of the original software or as you have revised it--and to make that easy, you can have access to the source code as well." However, one thing open source licenses tend to insist on is that the software may only be used pursuant to the conditions of the license. For example, the GPL states that nothing "grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License." [158] Such a statement may be rather too categorical. Even without accepting the license, one could copy, modify or distribute the software if the particular use qualified as fair use under section 107 of the copyright statute. [159]

Application of the fair use doctrine to open source software poses a number of puzzles. Fair use authorizes certain socially beneficial uses of copyrighted material that would otherwise be an infringement. [160] One who tapes a copyrighted program for future viewing, [161] or records a parody version of a copyrighted song, [162] or copies a scientific article in the course of research [163] may qualify for fair use. Fair use, however, is a notoriously vague doctrine. In deciding whether fair use applies, the court must consider four factors: the purpose of the use, the nature of the work, the amount used, and the effect on the market for licensing the work. [164]

Fair use issues could naturally arise both with open source developers copying from proprietary software and with others distributing open source software. The former case would be relatively straightforward, al though unpredictable, as fair use cases generally are. For example, an open source developer could copy some elements of a proprietary program to make her program interoperable with other software. If she took copyrighted elements (and because software is functional, much copying is not taking protected expression), the question of fair use would arise. The open source developer would argue for fair use along the usual factors--that the use was productive and noncommercial, that the copied work was functional and thus subject to a lower level of protection, that only so much was taken as was necessary, and that the open source project would not reduce the market for the product in question. Of course, in some cases, those arguments would not be available. The purpose of some open source projects is indeed to supplant commercial products. Similarly, fair use is likely to apply if an open source developer made a copy of a commercial product in order to reverse engineer it (i.e., to figure out how it works). [165] Although in a novel setting, such cases would not unduly tax the flexible structure of fair use analysis.

The more interesting question is the extent to which fair use applies to open source software itself. Many uses of open source software, of course, would not be challenged even if the user was not a party to an open source license. If someone made copies of some open source software, or modified the software, or distributed the software in a manner that was consistent with the license even if the user had not agreed to the license, then presumably the copyright holder would not object. This contrasts sharply, of course, with most commercial software where the copyright holder would likely object to any use that was not permitted pursuant to a licensing agreement. However, some uses of open source software might be objectionable to a developer who had set her software free. Uses that would not have been permitted under the license would now likely be objectionable. For example, if another developer incorporated open source code into a proprietary product where the license would have prohibited that or if someone distributed copies that failed to attribute authorship of the code correctly, then these uses would be considered objectionable.

The argument against fair use is a straightforward one. Open source licenses make software freely available and subject to very few restrictions, so there would seem to be little justification for permitting fair use. The most common justifications for fair use--that the copyright holder would not grant permission because of transaction costs or other obstacles to bargaining--are inapplicable.

One can construct a plausible argument for fair use, even where such use is inconsistent with the terms of the open source license. Take the example of a commercial developer who simply incorporates large portions of an open source program into some commercial software, and does not release the resulting product under an open source license, but rather sells it subject to typical commercial license that permits only limited uses of the executable code and no access to the source code. The first factor to consider is the nature of the use. This would not be research, teaching, or commentary--but simply a commercial use that weighs against fair use. However, other aspects of the use could turn the factor toward fair use. It could be productive use rather than merely reproductive--like making a copy, particularly if the code was modified. Where the user supplies independent creative input, fair use is more likely to apply.

Strangely enough, the very fact that the user was changing it from open source to commercial code could be used to argue for fair use. The unlikely analogy here would be to the rap music parody of the song "Pretty Woman" in Campbell v. Acuff-Rose Music. [166] The plaintiff in Campbell held the copyright in the venerable Roy Orbison song, "Pretty Woman." Defendants parodied the song in their own "Pretty Woman." [167] Defendants' parody effectively criticized the worldview of the "white-bread original." The essence of parody is that it must borrow from the original in order to ridicule it. The author of the original would likely refuse the use of his own work as a means for criticism; however, fair use plays a role here to further the deeper goals of copyright law--the creation and distribution of creative works. Where the author would use copyright to suppress expression (that is, to refuse permission to the making of a parody), fair use permits the use of copyrighted works to create works that comment upon them.

An analogy could also be made to the use of open source code in a commercial product. As with parody, the use is one that would not occur with the agreement of the copyright holder. The refusal to license arises out of a desire, in a very broad sense, to limit a species of expression. Certainly, the limitations arise out of the very best of intentions--the wish to keep the code solely in the open source arena and out of closed commercial products--but the same can be said of parody. The copyright holder may prefer not to hear a rap version of "Pretty Woman," or to see the photo of a pregnant movie star reproduced with the head of a comic actor, [168] or to have her "I Love New York" advertising campaign parodies in an "I Love Sodom" skit, [169] but fair use can be used to prevent such suppression of speech that is objectionable to the copyright holder. An argument can be made that copyright should not be used to control the expression of others. The counterargument, of course, is that the restrictions are necessary for open source to survive. If open source software could be readily used in commercial products, then the incentive to create open source software could be severely reduced. Under that view, the underlying purpose of copyright law, which is to promote creative expression, would be furthered by rejecting the analogy to parody.

The very nature of open source software also makes it more subject to fair use for another reason. Unpublished works are less likely to be subject to fair use. Thus, where The Nation printed excerpts from President Gerald Ford's soon-to-be-published autobiography, [170] and where a biography of J.D. Salinger included excerpts from his unpublished notes, [171] the courts denied fair use, despite the favored nature of the works in reporting historical facts and literary history. Commercial software has a peculiar status with respect to publication. The executable code is published by being distributed to the paying public, but the source code from which it springs is often kept unpublished. Accordingly, someone who managed to get some of the source code and publish it might not qualify for fair use, given the heightened protection for unpublished works. However, the whole point of open source software is to publish both the executable code and the source code, and as such, it would be more subject to fair use.

The final factor in fair use analysis is "the effect of the use upon the potential market for or the value of the copyrighted work." [172] Here again, the paradoxical nature of open source licenses--using intellectual property law to keep software free--leads to an inversion of the normal analysis. Suppose some open source code was incorporated, without permission, into a commercial program that performed a similar function (e.g., both programs were word-processing programs). The market analysis will naturally depend on the nature of the software, but it might be difficult for an open source copyright holder to show cognizable market harm. The obvious market harm would be the loss of potential licensing revenue, but if the code were freely available without a fee, then the loss of revenue would be zero. Another species of market harm could be the effect on the demand for the open source version. Some potential users of the open source version might instead choose the commercial version, which would constitute harm to the market; however, there are other ways to characterize it.

The commercial product arguably expands the market for the work. First, to the extent that the commercial product is successful, it could also increase demand for the open source version. Thus, although the use was contrary to the wishes of the copyright holder, she may not be able to show the sort of concrete market harm that courts require. Second, even if the commercial segment of the product market increases at the expense of the open source segment of the market, that may not constitute the sort of market harm that weighs against fair use. Where the issue is not loss of licensing revenue because the copyright holder has distributed her product for free, but rather control of a product market, the sort of loss at issue may not be protected from fair use. For example, in Sony Corp. of America v. Universal Studios, Inc., the holders of copyrights in television programs objected to the practice of home video taping of the programs. [173] Nevertheless, the Supreme Court held fair use applicable. [174] In part, the Court relied on the rationale that denying fair use would, in effect, grant broadcasters control over the market for videocassette recorders.

The foregoing analysis, suggesting that open source software may sometimes be more subject to fair use than commercial products, has a strange ring to it. Surely, if a software copyright holder has been so generous as to release her code subject only to the requirement that it remain open source, it would be perverse to allow that requirement to be ignored. However, the issue ultimately could be, what does copyright protect? If it is seen as a general form of property, then perhaps any restriction the copyright holder requires should be enforced. As discussed above, copyright can be seen as a balance. In order to promote the creation of works, authors are given a specific set of exclusive rights, but if the author effectively waives those rights and allows her code to be freely copied and distributed, but only with respect to some people, perhaps copyright law would not enforce her division of the market. Indeed, although the case law is presently scant, there is some authority for the proposition that using copyright to control a market would constitute misuse of copyright, although such use is better characterized as a more benign attempt to control a market than to keep it free.

This analysis would apply where the user is not a party to an open source license. Another set of issues arises with respect to how effective the open source license restrictions are on licensees. As noted, open source licenses are the most generous of legal documents--offering free access to copyrighted material. [175] However, open source licenses will typically have a couple of restrictions. For licenses like the venerable GPL, the most important provision, intended to ensure that the software remains open source, provides that the licensee cannot make the code private, or incorporate it into a product that is itself not open source. The license will also provide that if the licensee distributes adapted versions, she will make the adapted source code available. It may further provide that future versions will properly attribute the authors of the software.

Of course, those provisions will not have an effect on someone who is not a party to the license. So the open source licenses also have provisions addressing how one becomes a party to the license--and indeed, they are quite broad in some licenses. For example, the GPL states: "By modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it." [176] Such provisions would make agreement to a contract occur rather easily, going beyond even conventional clickwrap and shrinkwrap provisions. [177]

Other provisions in some open source licenses, if effective, would override any fair use rights that the licensee might have. Whether state licensing law should have such effect with respect to federal copyright privileges is, at best, unsettled. [178] Another provision common to open source licenses is a blanket exclusion of warranties and limitation of remedies, which is not a surprising allocation of risks, especially where such licenses are so frequently given without any charge. The license might also authorize the licensor to disable the software if there was a default in payment. [179]

So questions could arise as to the effectiveness of such license provisions under the foregoing analysis. [180] This concern has resulted in the drafting of a uniform law on licensing. [181] Because software companies were so active in the drafting process, this uniform law on licensing has been severely criticized as favoring licensors too heavily and giving little shrift to licensees. [182]

Without wading into the details of the proposed UCITA, [183] one can phrase the question generally: should such provisions as the ones cited above be made enforceable in order to strengthen the open source movement? Notably, this creates a strange alignment between the open source movement and its usual targets, proprietary software companies, because both would arguably benefit from a licensing law that gave broad powers to licensors. However, that alignment itself indicates the answer to the question. Many models of software distribution are likely to arise in the near future, from pure open source to the most closed proprietary system. To tilt licensing law in favor of licensors generally, or even to draft special provisions supporting some definition of an open source license, would likely cause more harm than good.

B. Direct Regulation of Software: of Dispersal, Monitoring, and Norms

Governments care quite a bit about what can be done with software. Some nations have made rather blunt attempts to control the information coming in and out of their respective countries via the Internet. [184] All governments want to regulate some aspects of computer networks--and want to regulate some potential uses very strictly. [185] Lessig makes the incisive point that open source software may be more difficult to regulate than proprietary software because the government would have to regulate not only the producer of the software, but also every other person who had a copy of the source code because each could modify the code. [186] This section looks to some other aspects of the interplay between open source licensing and government's ability to regulate code.

In some ways, the very openness of open source renders it more subject to government regulation. For example, suppose a government, for some reason, wished to prohibit the use of a particular algorithm. With respect to commercial, closed software, it might be difficult to determine whether the software used the algorithm. Examining the executable code and observing how the program behaved might not disclose whether the prohibited algorithm was used. This would depend on the nature of the algorithm, of course. Sometimes it would be quite obvious whether the program used the algorithm in question. But, other times, examination of the source code would be necessary. If the software company did not voluntarily turn the source code over for inspection, it might be necessary to obtain a warrant--which would require meeting the necessary standards of probable cause.

Open source, on the other hand, is freely accessible. The source code is freely distributed along with the executable code, so obtaining the source code would be quite easy. Thus, a governmental agency might have much greater ability to monitor and regulate open source software. Indeed, some open source licenses require that source code be made available to anyone who requests a copy, so an agency could simply request a copy, as opposed to entering an adversary proceeding to seek a warrant.

These legal distinctions might not make much difference in the real world. The terms of open source licenses might require that source code be readily available, but a licensee might not feel bound by the strict terms of the license. Even if refusing a request for the source code were a theoretical violation of the license, it would only matter if an action to enforce the license were brought. The government, as a nonparty, would presumably lack standing to enforce the license. The party who could enforce the license, the original open source licensor, might of course be quite unlikely to step in on behalf of the government to enforce a disclosure requirement against a fellow software developer. Even if that unlikely coalition were to arise, the remedy for the failure to disclose might not be an injunction requiring disclosure, but rather termination of the license.

Beyond enforcement, another aspect of regulability is politics. One might expect that open source would be more subject to regulation than strictly commercial interests. Public choice theory holds that broad, diffuse groups, like open source developers, are likely to be less effective in achieving political goals than narrow interest groups, such as the software industry. Indeed, congressional legislation in the intellectual property area has of late been driven largely by industry interests. However, outside the legislative sphere, open source may have more influence.

Open source might prove harder to regulate for purely political reasons, as the recent shift in United States policy on encryption illustrates. [187] For years, the United States has restricted export of encryption software. [188] In particular, even where the United States permitted export of certain encryption products, it permitted only export of the executable code, not the more useful source code. In other words, companies could export the functional executable code so that people could use some types of encryption, but not the transparent source code, which disclosed how it worked and could also be modified. Because other countries allowed export of more powerful encryption than the United States and because the efficacy of such restrictions is rather questionable, there was considerable industry pressure to relax the regulations. [189] The United States initially announced that more powerful encryption would be exportable, but again only in executable code form. Such rules would work heavily against open source developers, who depend on being able to deliver both executable and source code versions, and to share code across borders during development. The United States accordingly modified the proposed regulations to permit the export of source code as well. [190] In short, the moral suasion of open source developers achieved what insistent industry advocates had not. Thus, norms of software development may have influenced the regulation of software. [191]

Conclusion

Should intellectual property laws be amended or interpreted in order to foster open source? [192] Although openness is certainly a great virtue, [193] the experience until now counsels that legal theorists may be more limited than we realize in predicting the effects of changes in the law. The open source software offers many puzzles and lessons for intellectual property theory and doctrine. Perhaps the single lesson might be that engineers are smarter than lawyers. [194] Considerable legal scholarship in recent years has lamented the increases in intellectual property protection that have steadily diminished the public domain, but the open source movement has turned that process on its head. The open source movement has used strong protection of intellectual property to quite different ends. In particular, various open source licenses rest on strong copyright protection and restrictive licensing provisions. However, the open source licenses use such restrictive law to keep open source code free. Exactly because intellectual property laws place so much control in the hands of copyright owners, various flavors of open source licenses are able to finely tune the way in which code is kept available for others to study, modify, and redistribute. But, that does not mean that open source should provide a justification for strong intellectual property protection. The boom in software patents, for example, is a considerable threat to open source. [195]

Trying to interpret or enact intellectual property laws to foster open source may well be beyond the planning powers of lawyers, even if one were to assume that somehow such legislation would proceed free of the special interests that have guided so much legislation in the intellectual property area. Open source, in its most recent form, has built upon laws that seem to favor the opposite--restrictive intellectual property and licensing laws congenial to closed, proprietary models. Moreover, its very success will lead to many variations of open source. [196] Software companies are likely to make more of their source code accessible to the public, for various reasons: it can attract more developers to work with the product, it can increase the amount and quality of feedback from users, and it may establish the software as an industry standard. [197] A particularly interesting example could arise if Microsoft opens up some or all of its source code as part of a resolution of the antitrust case brought by the United States. [198] But such opening up of code will not always be so free as to qualify for the term "open source," as it is used now. In particular, companies may make source code accessible on more restrictive terms by limiting copying and modification, or by automatically transferring rights in modification to the original copyright holder. It is highly unlikely that lawmakers now envision all the permutations of open source that may arise, much less craft laws tailored to each one.

Another reason for concern springs from the government requiring disclosure of source code. Making software open source has undeniable social benefits, but the choice to make it open should lie with the author. Thus, there could be dangers in laws intended to encourage open source, because by definition, they would make legal benefits contingent on disclosing the source code. By contrast, the open source licensing model relies on voluntary participation, a much more congenial model.

FOOTNOTES:

n1 See, e.g., Open Sources: Voices From the Open Source Revolution (Chris DiBona et al., eds., 1999) (collection of essays on the history, theory and practice of open source software); on software law generally, see Mark Lemley et al., Software and Internet Law (Aspen Law & Business 2000). See also Mark A. Haynes, Black Holes of Innovation In the Software Arts, 14 Berkeley Tech. L.J. 567 (1999). The copyright listserve cni-copyright@cni.org, run by the Coalition for Networked Information, often has good discussion of both legal and social issues concerning open source software.

n2 Good places to start for information on Linux are http://www.linuxdoc.org/ (last modified Nov. 17, 2000) (the Linux Documentation Project) or http://www.linuxjournal.com/ (last modified Nov. 27, 2000) (Linux Journal).

n3 See http://mozilla.org/ (last visited Sept. 16, 2000) (Netscape's open source browser).

n4 See, e.g., Amy Harmon, A Surge in Popularity of Software that Unlocks the Code, N.Y. Times, Jan. 4, 1999, at C18.

n5 See http://www.gnu.org/ (last visited Sept. 16, 2000), or http://www.fsf.org/ (last visited Sept. 16, 2000) (home of the Free Software Foundation and the GNU project, source of some of the best-known pieces of free software and containing links to discussions of the philosophy behind free software). "Free software" is a better term than "open source" in some respects. This article uses "open source" simply for descriptive reasons, to focus on the legal implications of permitting access to the source code.

n6 See http://www.opencode.org/ (last visited Sept. 16, 2000) (consortium devoted to supporting the open code development model, associated with the Berkman Center for Internet & Society at Harvard Law School). The Berkman Center has also taken the open source approach in the litigation context with its Open Law project for pro bono litigation, in which it seeks to "develop arguments, draft pleadings, and edit briefs in public, online." See http://www.berkmancenter.org/ (last visited Sept. 16, 2000).

n7 See discussion infra Part II.

n8 Source code is "the form in which a computer program is written by the programmer. Source code is written in some formal programming language which can be compiled automatically into object code or executed by an interpreter." The Good Free Online Dictionary Of Computing at http://foldoc.doc.ic.ac.uk/foldoc/index.html (last visited Sept. 17, 2000).

n9 See id.

n10 There can be many variations on the source code/executable code distinction. Some computer languages, such as BASIC, are interpreted instruction by instruction. Java falls somewhere in between because it is not compiled into machinelevel executable code (which would limit it to running on one operating system), but rather into Java byte code, which in turn can be executed by a program called a Java virtual machine, running on any operating system.

n11 For a thorough discussion of the process of working with executable code, see Andrew Johnson-Laird, Software Reverse Engineering in the Real World, 19 U. Dayton L. Rev. 843 (1994).

n12 Where the licensee is dependent on the software, she may be concerned that the licensor will go out of business or, for some other reason, be unwilling or unable to modify the source code for future needs. In such settings, the parties often agree to put the source code in escrow, pending specified conditions. Such a transaction allows the licensor to maintain control over the source code while reassuring the licensee. See H. Ward Classen, Fundamentals of Software Licensing, 37 Idea 1 (1996); Nycum et al., Debugging Software Escrow: Will It Work When You Need It?, 4 Computer/L.J. 441 (1984); Viktoria L. Gres, Rejection of Computer Software Licensing Agreements in Bankruptcy, 8 Cardozo L. Rev. 361 (1986).

n13 If the developer relied completely on development tools, then perhaps she would not qualify as an "author" for copyright protection. See Joseph G. Arsenault, Software Without Source Code: Can Software Produced By A Computer Aided Software Engineering Tool Be Protected?, 5 Alb. L.J. Sci. & Tech. 131, 134 (1993).

n14 For a more detailed description of the process of producing software, see Lemley et al, supra note 1; see also Computer Assocs. Int'l, Inc. v. Altai, Inc., 982 F.2d 693 (2d Cir. 1992).

n15 See Dennis S. Karjala, The Relative Roles of Patent and Copyright in the Protection of Computer Programs, 17 J. Marshall J. Computer & Info. L. 41 (1998); Mark A. Lemley, Convergence in the Law of Software Copyright?, 10 High Tech L.J. 1 (1995); Mark A. Lemley & David W. O'Brien, Encouraging Software Reuse, 49 Stan. L. Rev. 255 (1997); David L. Hayes, What the General Intellectual Property Practitioner Should Know about Patenting Business Methods, 16 The Computer Law. 3 (Oct. 1999).

n16 See 17 U.S.C. § 103 (2000); 35 U.S.C. § 100 (2000).

n17 Jim Salter, A Practical Approach to Claiming Software, 14 Computer & High Tech. L.J. 435 (1998).

n18 Cf. Haynes, supra note 1, at 573 (noting that open source developers forgo the protections and incentives of patents).

n19 See 17 U.S.C. § 102 (2000).

n20 See Feist Publ'ns v. Rural Tel. Serv., 499 U.S. 340 (1991).

n21 See 17 U.S.C. § 411 (2000) (providing that a copyright must be registered for a domestic author to institute an infringement action and that statutory damages and attorney's fees are available only for infringement of a registered copyright).

n22 See 17 U.S.C. § 106 (2000). The application of those exclusive rights to digital technologies is not fully settled. See, e.g., I. Trotter Hardy, Computer "RAM" Copies: Hit or Myth?: Historical Perspectives on Caching as a Microcosm of Current Copyright Concerns, 22 U. Dayton L. Rev. 423, 427 (1997).

n23 See, e.g., 17 U.S.C. § 107 (2000).

n24 But see Rachel Chalmers, Code Critic, Salon Technology at http://www.salon.com/ /tech/feature/1999/11/30/lions/index.html (last visited Oct. 1, 2000) ("John Lions wrote the first, and perhaps only, literary criticism of UNIX, sparking one of open source's first legal battles.").

n25 See 37 C.F.R. § 202.20(c)(2)(vii)(A) (1999).

n26 See 37 C.F.R. § 202.20(c)(2)(vii)(A)(2) (1999).

n27 See id; see also 35 U.S.C. § 112 (1999).

n28 "As a general rule, where software constitutes part of a best mode of carrying out an invention, description of such a best mode is satisfied by a disclosure of the functions of the software. . . . Thus, flow charts or source code listings are not a requirement for adequately disclosing the functions of software." Fonar Corp. v. Gen. Elec. Co., 107 F.3d 1543, 1549 (Fed. Cir. 1997).

n29 Some have proposed that intellectual property laws be amended to require publication of source code. See Pamela Samuelson et al., A Manifesto Concerning the Legal Protection of Computer Programs, 94 Colum. L. Rev. 2308, 2427-29 (1994); see also Anthony J. Mahajan, Intellectual Property, Contracts, And Reverse Engineering After ProCD: A Proposed Compromise For Computer Software, 67 Fordham L. Rev. 329 (1999) (proposing that software producers be required to disclose their source code after a pre-determined passage of time as a condition to maintaining federal copyright and patent protection).

n30 See Alois Valerian Gross, Annotation, What is Computer "Trade Secret" Under State Law, 53 A.L.R. 4th 1046, 1055 (1987). Trade secret status is legally determined only through litigation, as distinguished from copyright or patent status. States have generally embraced the definition in either the Uniform Trade Secret Act § 1(4), 14 ULA 542 (1979) or Restatement of Torts § 757 (1939). Courts generally require that owners of trade secrets take security measures to safeguard trade secrets.

n31 Starting with Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240, 1248-51 (3d Cir. 1983), courts have consistently held software protected by copyright irrespective of whether it was in source code, executable code, or some other form. Nevertheless, there are good policy reasons to question whether copyright should apply to code in all its forms. See Pamela Samuelson, CONTU Revisited: The Case Against Copyright Protection for Computer Programs in MachineReadable Form, 1984 Duke L.J. 663, 705-49; Peter S. Menell, Tailoring Legal Protection for Computer Software, 39 Stan. L. Rev. 1329, 1371 (1987); A. Samuel Oddi, An Uneasier Case for Copyright than for Patent Protection of Computer Programs, 72 Neb. L. Rev. 351 (1993); Anthony L. Clapes, Confessions of an Amicus Curiae: Technophobia, Law, and Creativity in the Digital Arts, 19 U. Dayton L. Rev. 903 (1994) (arguing for higher level of copyright protection for software).

n32 Compare Vault Corp. v. Quaid Software Ltd., 655 F. Supp. 750 (E.D. La. 1987) and Green Book Int'l Corp. v. Inunity Corp., 2 F. Supp. 2d 112 (D. Mass. 1998) with S.O.S. Inc. v. Pauplay, Inc., 886 F.2d 1081 (9th Cir. 1989).

n33 Richard Stallman, Why Software Should Not Have Owners, at http://www.gnu.ai.mit/ .edu/philosophy/why-free.html (last visited Sept. 29, 2000).

n34 See id.

n35 For "a techie/hacker's case" for open source software, see The Case for Open Source: Hackers' Version, at http://www.opensource.org/for-hackers.html (last visited Sept. 29, 2000); for "a businessperson's case," see The Business Case for Open Source, at http://www.opensource.org/for-suits.html (last visited Sept. 29, 2000); for "a customer's case," see The

Customer Case for Open Source, at http://www.opensource.org/for-buyers.html (last visited Sept. 29, 2000).

n36 The Business Case for Open Source, at http://www.opensource.org/for-suits.html (last visited Sept. 29, 2000).

n37 See The Case for Open Source: Hackers' Version, at http://www.opensource.org/for-hackers.html (last visited Sept. 29, 2000).

n38 Think Free Speech, Not Free Beer, at http://www.opensource.walkerart.org/themarketf.html (last visited Sept. 29, 2000).

n39 For a list of open source licenses, see The Approved Licenses, at http://www.opensource.org/licenses/ (last visited Sept. 29, 2000) (listing The GNU General Public License (known as the GPL), the GNU Library or 'Lesser' Public License (LGPL), the BSD License; the MIT License; the Artistic License; the Mozilla Public License v. 1.0 (MPL); the Qt Public License (QPL). the IBM Public License; the MITRE Collaborative Virtual Workspace License (CVW License); the Ricoh Source Code Public License. the Python License; the zlib/libpng License; the Apache Software License, the Vovida Software License v. 1.0, the Sun Internet Standards Source License (SISSL), the Intel Open Source License, the Mozilla Public License 1.1 (MPL 1.1), and the Jabber Open Source License). The GPL, the "granddaddy" of open source licenses and still the most eloquent and thoughtful, covers among other things, the Linux operating system.

For a more general discussion of ways to share software, including open source, shareware, or public domain approaches, see Categories of Free and Non-Free Software, at http://www.gnu.ai.mit.edu/philosophy/categories.html (last visited Sept. 29, 2000).

n40 For an acute comparison of the legal effects of various open source licenses, see Frank Hecker, Setting Up Shop: The Business of Open-Source Software Revision 0.8, at http://www.hecker.org/writings/setting-up-shop.html (last visited Sept. 29, 2000).

n41 See id.; see also The Approved Licenses, at http://www.opensource.org/licenses/ (last visited Sept. 29, 2000).

n42 Bruce Perens, The Open Source Definition (Version 1.7), at http://www.opensource.org/

osd.html (last visited Nov. 25, 2000).

n43 See id.

n44 See id.

n45 See The Approved Licenses, at http://www.opensource.org/licenses (last visited Sept. 29, 2000).

n46 See Python 1.6, CNRI Open Source License Agreement, at http://www.handle.net/pythonlicenses/python1.69-5-00.html (visited Nov. 25, 2000).

n47 See Python 1.6, Beta 1 CNRI Open Source License Agreement, at http://www.handle.net/pythonlicenses/python1.6beta8-52000.html (last visited Sept. 29, 2000).

n48 See, e.g., Mozilla Public License Version 1.0, at http://www.mozilla.org/MPL/MPL-1.0.html (last visited Sept. 29, 2000).

n49 Or more precisely, the right to recover for patent infringement from anyone else that does. 35 U.S.C. § 271 (1994 & Supp. IV 1998).

n50 See, e.g., MIT License, at http://www.opensource.org/licenses/mit-licenses.html (last visited Oct. 12, 2000) ("Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software.").

n51 See 35 U.S.C. § 271 (1994 & Supp. IV 1998) (containing no mens rea requirement for infringement).

n52 A merchant who sells goods warrants that they are free of third party infringement claims. See U.C.C., Warranty of Title and Against Infringement, § 2-312(3) (2000) (A merchant seller "warrants that the goods shall be delivered free of the rightful claim of any third person by way of infringement."). Article 2, which covers sales of tangible goods, may not apply directly to software transactions, but is likely to be applied by analogy. See Andrew Beckerman-Rodau, Computer Software: Does Article 2 of the Uniform Commercial Code Apply?, 35 Emory L.J. 853 (1986).

n53 See U.C.C. § 2-312, 316 (2000) (providing that only merchants make automatic warranty of title and that warranties may be limited by circumstances, usage of trade).

n54 See, e.g., Wendy J. Gordon, An Inquiry into the Merits of Copyright: The Challenges of Consistency, Consent, and Encouragement Theory, 41 Stan. L. Rev. 1343 (1989) (comparing various theoretical bases for intellectual property law).

n55 On the application of economic analysis by legal scholars to specific copyright law doctrines, see Wendy J. Gordon, Fair Use As Market Failure: A Structural and Economic Analysis of the Betamax Case and Its Predecessors, 82 Colum. L. Rev. 1600 (1982). See also William M. Landes & Richard A. Posner, An Economic Analysis of Copyright Law, 18 J. Legal Stud. 325 (1989).

n56 See Mark A. Lemley & David McGowan, Legal Implications of Network Economic Effects, 86 Cal. L. Rev. 479, 528-30 (1998). See also Mark A. Lemley & David McGowan, Could Java Change Everything? The Competitive Propriety of a Proprietary Standard, 43 Antitrust Bull. 715, 716 (1998).

n57 See Lotus Dev. Corp. v. Borland Int'l, Inc., 49 F.3d 807, 814-15 (1st Cir.

1995) (holding menu command structure of spreadsheet program not protected by copyright), aff'd per curiam, 516 U.S. 233 (1996); see generally I. Trotter Hardy, Copyright And "New-Use" Technologies, 23 Nova L. Rev. 659 (1999); Niva Elkin-Koren, Copyrights In Cyberspace--Rights Without Laws?, 73 Chi.-Kent L. Rev. 1155 (1998); see also Grace H. Lee, The Copyrightability Of Computer Software, 6 U. Balt. Intell. Prop. L.J. 117 (1998).

n58 See Lemley & McGowan, Legal Implications of Network Economic Effects, supra note 56, at 527-30.

n59 See id.

n60 See id.

n61 See id.

n62 See id.

n63 See id.

n64 See id. (Some operating systems in the past have included the capacity to run Microsoft-compatible software.).

n65 A somewhat different issue of economic incentives arises in the case of nonprofit educational institutions and intellectual property rights in software. See generally J. H. Reichman, Computer Programs As Applied Scientific Know-How: Implications of Copyright Protection for Commercialized University Research, 42 Vand. L. Rev. 639 (1989).

n66 See http://www.winehq.com/ (site for the windows emulator project); Eben Moglen, Bill Gates' Best Bet is to Set Software Free, San Jose Mercury News, Dec. 26, 1999.

n67 U.S. Const. art. I, § 8, cl. 8.

n68 Id. On economics and intellectual property generally, see Kenneth W. Dam, Some Economic Considerations in the Intellectual Property Protection of Software, 24 J. Legal Stud. 321 (1995). For a discussion of economic analysis of computer technology law, see Seth A. Cohen, To Innovate or Not to Innovate, That is the Question: The Functions, Failures, and Foibles of the Reward Function Theory of Patent Law in Relation to Computer Software Platforms, 5 Mich. Telecomm. & Tech. L. Rev. 1 (1998).

n69 Mark A. Lemley, The Economics of Improvement in Intellectual Property Law, 75 Tex. L. Rev. 989 (1997).

n70 See, e.g., Sony Corp. of Am. v. Universal City Studios, Inc., 464 U.S. 417, 429 (1984); Kewanee Oil Co. v. Bicron Corp., 416 U.S. 470, 480 (1974).

n71 See 17 U.S.C. § 102 (1994) (extending copyright to all original works of authorship fixed in tangible form).

n72 See 17 U.S.C. § 302 (1994).

n73 See Digital Millennium Copyright Act of 1998, 17 U.S.C. § 1201 (1998).

n74 74 F. Supp 2d. 1 (D.D.C. 1999). Appropriately enough, the plaintiff in Eldred is being represented pro bono by the Openlaw project at Harvard Law School. See http://eon.law.harvard.edu/opencode (last visited Nov. 28, 2000).

n75 Pub. L. No. 105-298, 112 Stat. 2827 (codified at 17 U.S.C. § 304(b) (2000)).

n76 See, e.g., Grant S. Nelson & Robert J. Pushaw, Jr., Rethinking the Commerce Clause: Applying First Principles to Uphold Federal Commercial Regulations but Preserve State Control Over Social Issues, 85 Iowa L. Rev. 1, 79-81 (1999) (describing Supreme Court jurisprudence broadening Congress' Commerce Clause powers).

n77 The trial court ruled against plaintiff and the case is currently on appeal. See Eldred, 74 F. Supp. 2d at 4.

n78 Id. at 3.

n79 See Ruth Gana Okediji, Copyright and Public Welfare in Global Perspective, 7 Ind. J. Global Leg. Stud. 117 (1999) (criticizing economic arguments behind international movement toward expansion of intellectual property rights); Lakshmi Sarma, Comment, Biopiracy: Twentieth Century Imperialism In The Form Of International Agreements, 13 Temp. Int'l & Comp. L.J. 107 (1999).

n80 See, e.g., Paul Goldstein, Copyright's Highway: The Law and Lore of Copyright From Gutenberg to the Celestial Jukebox (1994); Tom W. Bell, Fair Use v. Fared Use: The Impact of Automated Rights Management on Copyright's Fair Use Doctrine, 76 N.C. L. Rev. 557 (1998); I. Trotter Hardy, Property (and Copyright) in Cyberspace, 1996 U. Chi. Legal F. 217; Robert P. Merges, Contracting into Liability Rules: Intellectual Property Rights and Collective Rights Organizations, 84 Cal. L. Rev. 1293 (1996). See also Information Infrastructure Task Force, Intellectual Property and the National Information Infrastructure: The Report of the Working Group on Intellectual Property (1995) (federal report proposing changes to copyright laws reflecting the expansive propertization approach); Ann Okerson, Who Owns Digital Works?, Sci. Am. (July 1, 1996). On the more general tendency of property rights to expand along with a global trend toward privatization, see Carol M. Rose, The Several Futures of Property: Of Cyberspace and Folk Tales, Emission Trades and Ecosystems, 83 Minn. L. Rev. 129 (1998). On the diminishment of the public domain by propertization of ideas, see David Lange, Recognizing the Public Domain, 44 Law & Contemp. Probs. 147 (1981).

n81 See James Boyle, Shamans, Software, and Spleens: Law and the Construction of the Information Society (1996); Julie E. Cohen, Lochner in Cyberspace: The New Economic Orthodoxy of "Rights Management", 97 Mich. L. Rev. 462 (1998) (refuting a number of theoretical bases for maximal copyright protection); Yochai Benkler, Free As The Air To Common Use: First Amendment Constraints on Enclosure of the Public Domain, 74 N.Y.U. L. Rev. 354 (1999); Lydia Pallas Loren, Redefining The Market Failure Approach to Fair

Use in an Era of Copyright Permission Systems, 5 J. Intell. Prop. L. 1; Maureen A. O'Rourke, Fencing Cyberspace: Drawing Borders in a Virtual World, 82 Minn. L. Rev. 609 (1998). See also Margaret Jane Radin & R. Polk Wagner, On the Internet and Legal Theory: The Myth of Private Ordering: Rediscovering Legal Realism in Cyberspace, 73 Chi.-Kent. L. Rev. 1295 (1998); Margaret Jane Radin, Property Evolving in Cyberspace, 15 J.L. & Com. 509 (1996) (discussing how alleged benefits of privatizing information may be outweighed by unintended consequences); Pamela Samuelson, The Copyright Grab, Wired, Jan. 1996, at 134; Michael J. Meurer, Price Discrimination, Personal Use and Piracy: Copyright Protection of Digital Works, 45 Buff. L. Rev. 845 (1997). For a general discussion of the importance of maintaining a robust public domain, see Jessica Litman, The Public Domain, 39 Emory L.J. 965 (1990); see also Rosemary J. Coombe, Objects of Property and Subjects of Politics: Intellectual Property Laws and Democratic Dialogue, 69 Tex. L. Rev. 1853 (1991) (discussing adverse effects of commodification of cultural information).

n82 There are many calls for minimal or zero intellectual property protection. The most comprehensive economic analysis of law supporting what the text above calls the spartan approach is Glynn S. Lunney, Jr., Trademark Monopolies, 48 Emory L.J. 367 (1999); see also Glynn S. Lunney, Jr., Reexamining Copyright's Incentives-Access Paradigm 1996, 49 Vand. L. Rev. 483 (1996).

n83 See Lunney, Reexamining Copyright's Incentives-Access Paradigm, supra note 82, at 556-61.

n84 See id. at 491.

n85 See also Richard Stallman, Reevaluating Copyright: The Public Must Prevail, 75 Or. L. Rev. 291 (1996).

n86 See Lunney, Trademark Monopolies, supra note 82, at 371-72. For a general criticism of the recent expansion of trademark protection and its propertization rationale, see Mark A. Lemley, The Modern Lanham Act and the Death of Common Sense, 108 Yale L.J. 1687 (1999).

n87 See Lunney, Trademark Monopolies, supra note 82.

n88 See Richard H. McAdams, Relative Preferences, 102 Yale L.J. 1 (1992).

n89 See id. at 25.

n90 See id.

n91 See infra Part IV.

n92 See Robert Merges, Contracting into Liability Rules, 84 Cal. L. Rev. 1293, 1334-35 (1996).

n93 See Neil Netanel, Copyright Alienability Restrictions and the Enhancement of Author Autonomy: A Normative Evaluation, 24 Rutgers L.J. 347, 365 (1993) (distinguishing United States and continental European approaches); see also Roberta Kwall, Copyright and the Moral Right: Is an

American Marriage Possible?, 38 Vand. L. Rev. 1 (1985) (discussing the interplay between American Copyright and the traditional European/Third World moral right doctrine).

n94 Universal Declaration of Human Rights, Art. 27, at http://www.un.org/ /rights/50/decla.htm.

n95 See Justin Hughes, The Philosophy of Intellectual Property, 77 Geo. L.J. 287, 299 (1988); cf. Wendy J. Gordon, A Property Right in Self-Expression: Equality and Individualism in the Natural Law of Intellectual Property, 102 Yale L.J. 1533, 1560-72 (1993) (discussing Locke's theory of property).

n96 Hughes, supra note 95, at 300; see also Gordon, supra note 95; Adam D. Moore, A Lockean Theory Of Intellectual Property, 21 Hamline L. Rev. 65 (1997).

n97 See generally Margaret Radin, Property and Personhood, 34 Stan. L. Rev. 957 (1982) (discussing the relationship between property and personhood).

n98 See Lunney, Trademark Monopolies, supra note 82.

n99 See id.

n100 Julie C. Smith, The NII Copyright Act of 1995: A Roadblock Along the Information Superhighway, 8 Seton Hall Const. L.J. 891, 939 (1998); Alan S. Gutterman, The NorthSouth Debate Regarding the Protection of Intellectual Property Rights, 28 Wake Forest L. Rev. 89, 109 (1993).

n101 See Alexander Gigante, Ice Patch on the Information Superhighway: Foreign Liability For Domestically Created Content, 14 Cardozo Arts & Ent. L.J. 523, 532 (1996); Craig A. Wagner, Motion Picture Colorization, Authenticity, and the Elusive Moral Right, 64 N.Y.U. L. Rev. 628, 707 (1989).

n102 See Visual Artists Rights Act, Pub. L. 101-650, Title VI, Dec.1, 1990, 104 Stat. 5128 (codified at 17 U.S.C. § 106 (1994))

n103 Electronic Numerical Integrator and Computer.

n104 See Scott Mccartney, ENIAC: The Triumphs and Tragedies of the World's First Computer (Walker & Co. 1999).

n105 Id.

n106 Cf. John A. Gibby, Software Patents: A Programmer's Perspective, 23 Rutgers Computer & Tech L.J. 293 (1997).

n107 See High-Wire Act for Science, L.A. Times, Jan. 9, 2000, at M4.

n108 See Nicholas Wade, Scientist at Work: Joe Z. Tsien; Of Smart Mice and an Even Smarter Man, N.Y. Times, Sept. 7, 1999, at F1.

n109 See id.

n110 See Diamond v. Diehr, 450 U.S. 175 (1981); Michael L. Kiklis, The Demise of the Mathematical Algorithm Rejection and the Emergence of the Utility-Based Section 101 Inquiry, 16 The Computer Law. 26 (Aug. 1999).

n111 See Gottschalk v. Benson, 409 U.S. 63 (1972); see also John Fellas, The Patentability of Software-related Inventions in the United States, 21 Eur. Intell. Prop. Rev. 330 (1999).

n112 See Gottschalk, 409 U.S. at 71-72.

n113 See Parker v. Flook, 437 U.S. 584 (1978).

n114 See Diamond v. Diehr, 450 U.S. 175 (1981).

n115 See AT&T Corp. v. Excel Communications Inc., 175 F.3d 1352 (Fed. Cir. 1999) (holding patent protection broadly available for software as process as well as machine); State Street Bank & Trust Co. v. Signature Fin. Group, 149 F.3d 1368, (Fed. Cir. 1998), cert. denied, 525 U.S. 1093 (1999) (upholding patentability for software implementing business method, which implemented mutual fund plan); In re Alappat, 33 F.3d 1526 (Fed. Cir. 1994); In re Lowry, 32 F.3d 1579 (Fed. Cir. 1994). See also In re Freeman, 573 F.2d 1237 (C.C.P.A. 1978); In re Walter, 618 F.2d 758 (C.C.P.A. 1980); and In re Abele, 684 F.2d 982 (C.C.P.A. 1982) (early trilogy of Court of Customs and Patent Appeals cases attempting to formulate rules for patentability of software). The United States cases have had a certain amount of persuasive weight in other countries. See, e.g., Natalie Stoianoff, Patenting Computer Software: An Australian Perspective, 21 Eur. Intell. Prop. Rev. 500 (1999) (discussing use of U.S. precedent in Australia); Richard H. Stern, Scope-of-Protection Problems with Patents and Copyrights on Methods of Doing Business, 10 Fordham Intell. Prop. Media & Ent. L.J. 105 (1999); Christopher S Cantzler, Comment, Leading the Way to Consistency for Patentability of Computer Software, 71 U. Colo. L. Rev. 423 (2000); Robert A. Kreiss, Patent Protection for Computer Programs and Mathematical Algorithms: the Constitutional Limitations on Patentable Subject Matter, 29 N.M. L. Rev. 31(1999). The free issuance of software patents may even lead practitioners to forgo means to draft patents that will stand up in litigation. See Michael L. Kiklis, A Patent Saved is a Patent Earned, 17 The Computer Law. 3 (Jan. 2000); Examination Guidelines for Computer-Related Inventions, 61 Fed. Reg. 7478 (1996); Wesley L. Austin; Software Patents, 14 The Computer Law. 14 (June 1997); Keith Stephens, Software Patent Developments: The PTO's Examination Guidelines for Computer-Related Inventions, 17 J. Marshall J. Computer & Info. L. 277 (1998). On the continuing struggle to define the subject matter of patents in new technologies, see R. Carl Moy, Statutory Subject Matter And Hybrid Claiming, 17 J. Marshall J. Computer & Info. L. 277 (1998).

n116 Michael D. McCoy, Patents.Com: Exclusivity for ECommerce, 16 The Computer Law. 10 (Dec. 1999) (discussing a number of notable business method patents).

n117 "There are three general ways software can be used in an invention to satisfy the patentable subject matter requirement. First, software is deemed a 'machine' when a computer is included in the claims of the invention. Second, software is a 'process' when it is claimed as a series of operational steps to be performed on or with the aid of a computer. The computer is not part of the invention as it is with the 'machine' classification. And finally, a computer readable memory (e.g., floppy disks, CDROM, system memory) that can be used to direct a computer to function in a particular manner when used by the computer is considered an 'article of manufacture.' Consequently, software in various forms constitutes a machine, process, or an article of manufacture, depending on how it is claimed." Nora M. Tocups & Robert J. O'Connell, Patent Protection for Computer Software, 14 The Computer Law. 18 (November 1997) (quoting In re Beauregard, 53 F.3d 1583, 1584 (Fed. Cir. 1995)). See Michael J. Mehrman, Strategic Concerns when Pursuing Foreign Patents in the Computer Arts, 15 The Computer Law. 17 (March 1998) (discussing differing levels of patent protection in United States and Europe); J.H. Reichman, Legal Hybrids Between the Patent and Copyright Paradigms, 94 Colum. L. Rev. 2432(1994); Michael J. Klein, Software Patenting: A New Approach, 6 U. Balt. Intell. Prop. L.J. 135 (1998); see also Vincent Chiappetta, Patentability of Computer Software Instruction as an "Article of Manufacture:" Software as Such as the Right Stuff, 17 J. Marshall J. Computer & Info. L. 89 (1998).

n118 See Jeffrey R. Kuester et al., A New Frontier in Patents: Patent Claims to Propagated Signals, 17 J. Marshall J. Computer & Info. L. 75 (1998).

n119 See, e.g., Richard Stallman, The GNU Operating System and the Free Software Movement, in Open Sources: Voices From the Open Source Revolution 67 (Chris DiBona et al, eds., 1999) ("The worst threat we face comes from software patents, which can put algorithms and features off-limits to free software for up to twenty years.").

n120 Simon Garfinkel, Patently Absurd, Wired 2.07.

n121 This point is made in discussions of open source and patents on slashdot.com and the cni-copyright discussion list.

n122 On the special problems of applying nonobviousness analysis in new technological areas, see John Kaskan, Obviousness and New Technologies, 10 Fordham Intell. Prop. Media & Ent. L.J. 159 (1999); see also Qing Lin, A Proposed Test for Applying the Doctrine of Equivalents to Biotechnology Inventions: The Nonobviousness Test, 74 Wash. L. Rev. 885 (1999); A. Samuel Oddi, Beyond Obviousness: Invention Protection in the Twenty-First Century, 38 Am. U. L. Rev. 1097 (1998). On nonobviousness generally, see Nonobviousness: The Ultimate Condition of Patentability (John F. Witherspoon, ed. 1980).

n123 See 35 U.S.C. § § 103(c) 102(e-g) (1999).

n124 See, e.g., Tocups & O'Connell, supra note 117, at n.2021; see also Seth Shulman, Software Patents Tangle the Web, Tech. Rev. (Mar./Apr. 2000), at http://www.techreview/. com/articles/ma00/shulman.htm.

n125 See, e.g., Philippe G. Ducor, Patenting the Recombinant Products of Biotechnology and Other Molecules 15 (Kluwer Law Int. 1998) (stating that novelty analysis in biotech "is generally not difficult to evaluate").

n126 See Gref Ahorian, Internet Patent New Service (Feb. 10, 2000) at http://bustpatents.com/archive.htm.

n127 In addition to the USPTO, the Software Patent Institute is attempting to build resources for searching prior art in the software area.

n128 See Privacy Software Patent may be Challenged by Web Protocol Developers, 58 BNA Patent, Trademark & Copyright J. 284.

n129 See Patent Barred For Compton's, The New York Times, Oct. 31, 1994, at D7.

n130 See Privacy Software Patent may be Challenged by Web Protocol Developers, supra note 128; at 284; Frederick H. Colen & Robert D. Kucler, Re-exam of Y2K Patent: Much at Stake, Nat'l L. J., Mar. 13, 2000, at B10.

n131 See id.

n132 See Arti K. Rai, Intellectual Property Rights in Biotechnology: Addressing New Technology, 34 Wake Forest L. Rev. 827 (1999).

n133 See Mccartney, supra note 104.

n134 On some of the tangled issues in patent litigation law, see John R. Thomas, On Preparatory Texts And Proprietary Technologies: The Place Of Prosecution Histories in Patent Claim Interpretation, 47 UCLA L. Rev. 183 (1999); Rafael X. Zahralddin, Note, The Effect Of Broad Patent Scope on the Competitiveness of United States Industry, 17 Del. J. Corp. L. 949 (1992).

n135 See Steven C. Carlson, Note, Patent Pools and The Antitrust Dilemma, 16 Yale J. on Reg. 359 (1999).

n136 On the changing jurisprudence of the doctrine of equivalents, see Scott R. Boalick, Note, The Dedication Rule and the Doctrine of Equivalents: A Proposal for Reconciliation, 87 Geo. L.J. 2363 (1999).

n137 See David J. Franklyn, The Apparent Manufacturer Doctrine, Trademark Licensors and the Third Restatement of Torts, 49 Case W. Res. L. Rev. 671, 692-96 (1999).

n138 See Underwriter's Labs., Inc. v. Smith, 246 N.Y.S.2d 436 (N.Y. Sup. Ct. 1964); see also John V. Tait, Trademark Regulations and the Commercial Speech Doctrine: Focusing on the Regulatory Objective to Classify Speech for First Amendment Analysis, 67 Fordham L. Rev. 897, 900-01 (1998).

n139 See Perens, supra note 42.

n140 I note that the term "open source" is apparently already in use in the vernacular of intelligence-gathering to mean publicly available information. See, e.g., Leslie A. Benton & Glenn T. Ware, Haiti: A Case Study of the International Response and the Efficacy of Nongovernmental Organizations in the Crisis, 12 Emory Int'l L. Rev. 851 (1998) ("to collect and process opensource and J2-provided intelligence related to civil-military and civic action operations.").

n141 Cf. Jitka Smith, Comment, Budweiser or Budweiser?, 32 J. Marshall L. Rev. 1251 (1999).

n142 A descriptive mark may be registrable if it acquires "secondary meaning." See, e.g., Volkswagenwerk Aktiengesellschaft v. Rickard, 492 F.2d 474, 477 (5th Cir. 1974).

n143 See Eric S. Raymond, Issued by and for the Board of Directors of OSI, 16 June 1999 Announcement of "OSI Certified" open source mark, at http://www.opensource.org/

pressreleases/certified-open-source.html (last visited Nov. 25, 2000).

n144 For discussions of the original meaning of "hacker," and for many examples of the contributions of such hackers to the plasticity of English, see The New Hacker's Dictionary (Eric Raymond, compiler, 3d ed. 1996).

n145 See David G. Post, Governing Cyberspace, 43 Wayne L. Rev. 155 (1996); David R. Johnson & David Post, Law and Borders -The Rise of Law in Cyberspace, 48 Stan. L. Rev. 1367 (1996); Greg Y. Sato, Should Congress Regulate Cyberspace?, 20 Hastings Comm. & Ent. L.J. 699 (1998); Sean Selin, Governing Cyberspace: The Need for an International Solution, 32 Gonz. L. Rev. 365 (1997); Robert Norman Sobol, Intelligent Agents and Futures Shock: Regulatory Challenges of the Internet, 25 Iowa J. Corp. L. 103 (1999) (book review).

n146 Lawrence Lessig, Code and Other Laws of Cyberspace (1999).

n147 Id.

n148 See I. Trotter Hardy, The Proper Legal Regime for "Cyberspace", 55 U. Pitt. L. Rev. 993 (1994).

n149 See Lessig, supra note 146.

n150 See Michael J. Madison, Legal-Ware: Contract And Copyright In The Digital Age, 67 Fordham L. Rev. 1025 (1998). On the international ramifications of licensing law, see Kalama Lui-Kwan & Kurt Opsahl, The Legal and Policy Framework for Global Electronic Commerce: A Progress Report, 14 Berkeley Tech. L.J. 503 (1999).

n151 U.C.C. § 2.

n152 See Mark Lemley, Intellectual Property and Shrinkwrap Licenses, 68 S. Cal. L. Rev. 1239 (1995); see also Charles R. McManis, The Privatization (or "Shrink-Wrapping") of American Copyright Law, 87 Cal. L. Rev. 173 (1999).

n153 See Maureen A. O'Rourke, Drawing the Boundary Between Copyright and Contract: Copyright Preemption of Software License Terms, 45 Duke L.J. 479 (1995); ProCD, Inc. v. Zeidenberg, 86 F.3d 1447 (7th Cir. 1996).

n154 See Holly K. Towle, The Politics Of Licensing Law, 36 Hous. L. Rev. 121 (1999).

n155 See id.

n156 See, e.g., Pamela Samuelson & Kurt Opsahl, Licensing Information in the Global Information Market: Freedom of Contract Meets Public Policy, 21 Eur. Intell. Prop. Rev. 386 (1999) (discussing how draft 2B could upset the delicate balance of intellectual property law and antitrust law); A. Michael Froomkin, Article 2B as Legal Software for Electronic Contracting -Operating System or Trojan Horse?, 13 Berkeley Tech. L.J. 1023 (1998).

n157 The proposed UCITA is available at http://www.ucita.org/ (last visited Nov. 27, 2000). See also Pratik A. Shah, The Uniform Computer Information Transactions Act, 15 Berkeley Tech. L.J. 85 (2000).

n158 See GNU General Purpose License, at http://www.gnu.org/ (last visited Nov. 28, 2000).

n159 See William W. Fisher III, Reconstructing the Fair Use Doctrine, 101 Harv. L. Rev. 1659 (1988).

n160 See 17 U.S.C. § 107 (1994).

n161 See Sony Corp. v. Universal City Studios, Inc., 464 U.S. 417 (1984)

n162 See Margaret E. Watson, Unauthorized Digital Sampling in Musical Parody: a Haven in the Fair Use Doctrine?, 21 W. New Eng. L. Rev. 469 (1999).

n163 But perhaps not if done on a systematic basis. See Am. Geophysical Union v. Texaco, 60 F.3d 913 (2d Cir. 1995).

n164 See 17 U.S.C. § 107 (1994).

n165 See Sony Computer Entm't, Inc. v. Connectix Corp., No. 99-15852 (9th Cir. Feb. 10, 2000).

n166 See Campbell v. Acuff-Rose Music, 510 U.S. 569 (1994).

n167 See id.

n168 See Leibovitz v. Paramount Pictures, 137 F.3d 109 (2d Cir. 1998).

n169 See Campbell v. Acuff-Rose Music, 510 U.S. 569 (1994) (citing Elsmere Music, Inc. v. Nat'l Broad. Co., 623 F.2d 252 (2d Cir. 1980)).

n170 See Harper & Row, Publishers, Inc. v. Nation Enters., 471 U.S. 539 (1985).

n171 See Salinger v. Random House, Inc., 811 F.2d 90 (2d Cir. 1987).

n172 Sony Corp. of Am. v. Universal City Studios, Inc., 464 U.S. 417, 451

(1983).

n173 See Sony Corp. of Am. v. Universal City Studios, Inc., 464 U.S. 417 (1983).

n174 See id.

n175 See supra Part I.

n176 http://www.gnu.org/copyleft/gpl.html (last visited Nov. 28, 2000).

n177 Garry L. Founds, Note, Shrinkwrap and Clickwrap Agreements: 2B or not 2B?, 52 Fed. Comm. L.J. 99 (1999).

n178 See Vault Corp. v. Quaid Software Ltd., 847 F.2d 255 (5th Cir. 1988); Julie E. Cohen, Reverse Engineering and the Rise of Electronic Vigilantism: Intellectual Property Implication of "Lock-Out" Programs, 68 S. Cal. L. Rev. 1091 (1995); David A. Rice, Public Goods, Private Contract and Public Policy: Federal Preemption of Software License Prohibitions Against Reverse Engineering, 53 U. Pitt. L. Rev. 543 (1992); Apik Minassian, Comment, The Death of Copyright: Enforceability of Shrinkwrap Licensing Agreements, 45 UCLA L. Rev. 569 (1997).

n179 See Kenneth W. Dam, Self-Help In The Digital Jungle, 28 J. Legal Stud. 393 (1999).

n180 See Mark A. Lemley, Beyond Preemption: The Law and Policy of Intellectual Property Licensing, 87 Cal. L. Rev. 111 (1999); Robert W. Gomulkiewicz, The License Is The Product: Comments on the Promise of Article 2B for Software and Information Licensing, 13 Berkeley Tech. L.J. 891 (1998); William W. Fisher III, Property And Contract On The Internet, 73 Chi.-Kent L. Rev. 1203 (1998); Wendy J. Gordon, Intellectual Property as Price Discrimination: Implications for Contract, 73 Chi.-Kent L. Rev. 1367 (1998).

n181 See David A. Rice, Digital Information as Property & Product: U.C.C. Article 2B, 22 U. Dayton L. Rev. 621 (1997); Julie E. Cohen, Copyright and the Jurisprudence of Self-Help, 13 Berkeley Tech. L.J. 1089 (1998); Raymond T. Nimmer, Breaking Barriers: The Relation Between Contract and Intellectual Property Law, 13 Berkeley Tech L.J. 827 (1998); David McGowan, Free Contracting, Fair Competition, and Draft Article 2B: Some Reflections on Federal Competition Policy, Information Transactions, and "Aggressive Neutrality", 13 Berkeley Tech L.J. 1173 (1998).

n182 See, e.g., Rochelle Cooper Dreyfuss, Do You Want to Know a Trade Secret? How Article 2B Will Make Licensing Trade Secrets Easier (But Innovation More Difficult), 87 Cal. L. Rev. 191 (1999); see also Niva Elkin-Koren, Copyright Policy and the Limits of Freedom of Contract, 12 Berkeley Tech L.J. 93 (1997).

n183 The proposed UCITA is available at http://www.ucita.org/ (last visited November 27, 2000).

n184 See, e.g., Germany v. Somm, 1 ILR (P&F) 832 (Munich Local Court July

15, 1998).

n185 See A. Michael Froomkin, Of Governments And Governance, 14 Berkeley Tech L.J. 617 (1999) (discussing topdown governance, such as that imposed by governments, versus bottom-up governance, such as that represented by voluntary participation in the open source movement); see also David Goldstone & Betty-Ellen Shave, International Dimensions Of Crimes In Cyberspace, 22 Fordham Int'l L.J. 1924 (1999); Marc A. Moyer, Section 301 of the Omnibus Trade and Competitiveness Act of 1988: A Formidable Weapon in the War Against Economic Espionage, 15 J. Int'l L. Bus. 178 (1994).

n186 See Lawrence Lessig, The Limits in Open Code: Regulatory Standards and the Future of the Net, 14 Berkeley Tech. L.J. 759 (1999); see also Lawrence Lessig, Open Code and Open Societies: Values of Internet Governance, 74 Chi.-Kent. L. Rev. 1405 (1999).

n187 Cf. Porter Goss, An Introduction to the Impact of Information Technology on National Security, 9 Duke J. Comp. & Int'l L. 391 (1999).

n188 See 50 U.S.C.S. App. § 2403 (2000); see also Export Administration Regulations, at http://w3.access.gpo.gov/bxa/.

n189 See Maureen S. Dorney, The Grip on Encryption: Export limits on cyphering technology are strangling U.S. software companies, but relief may be on the way, at http://www.ipmag.com/dorney.html (visited Nov. 25, 2000).

n190 The case also has constitutional dimensions; source code is more likely than executable code to be accorded First Amendment protection as speech. See Universal City Studios v. Reimerdes, No. 00 Civ. 0277(LAK) (S.D.N.Y. Feb. 2, 2000) (holding that executable code at issue did "little to further traditional First Amendment interests" because its expressive content was minimal compared to its functional component). Whether source code is protected speech is at present unsettled. See Lawrence Lessig, The Law of the Horse: What Cyberlaw Might Teach, 113 Harv. L. Rev. (1999) (discussing cases differing on whether source code for encryption software is protected speech); Encryption/First Amendment, Ninth Circuit Holds That Export Administration Regulations Violate First Amendment, 16 The Computer Law.; The SoftSpeech Discussion List's Web Pages, at http://samsara.law.cwru.edu/sftspch/ (collecting resources on the issue of whether source code is protected speech).

n191 See Mark A. Lemley, The Law And Economics Of Internet Norms, 73 Chi.-Kent L. Rev. 125 (1998).

n192 See Neil Weinstock Netanel, Copyright and a Democratic Civil Society, 106 Yale L.J. 283 (1996); Niva ElkinKoren, Cyberlaw and Social Change: A Democratic Approach to Copyright Law in Cyberspace, 14 Cardozo Arts & Ent. L.J. 215 (1996); for a summary of recent work on information law and its social effects, See Keith Aoki, Innovation and the Information Environment: Interrogating the Entrepreneur, 75 Or. L. Rev. 1 (1996). On the role of intellectual property law in the coming networked world, see Intellectual Property in the Age of Universal Access (Pamela Samuelson & Peter G. Neumann eds., 1999).

n193 See Lessig, supra note 146, (providing a far-sighted analysis of the social benefits of open code). See also Open Sources, supra note 1.

n194 A different issue is the idea of fostering development of the software industry itself. See, e.g., Helene Miale, Note, The National Competitiveness Act: Gauging The Federal Government's Role in Promoting Technology Policy To Enhance U.S. Economic Growth, 18 Seton Hall Legis. J. 779 (1994).

n195 See supra notes 118-119, and accompanying text. See also Bryan Pfaffenberger, The Coming Software Patent Crisis: Can Linux Survive?, Linux Journal (1999).

n196 For instance, Sun Microsystems has opened up the source code to its Solaris operating system software, but retained restrictions in the license that would make it fail to qualify for the "OSI Certified: certification mark. See Hiawatha Bray, Linux Leaders are Cool to Sun Source-Code Plan, The Boston Globe Oct. 2, 1999, at F1. See also Steve Hamm, The Wild And Woolly World Of Linux, Bus. Week, Nov. 15, 1999, Information Technology section (describing various combinations of proprietary and open source software).

n197 See Benkler, supra note 81, at 404.

n198 See United States v. Microsoft Corp., 97 F. Supp. 2d 59 (D.D.C. 2000).