THE TAO OF OPEN SOURCE: MINIMUM ACTION FOR MAXIMUM GAIN*

By Matthew D. Satchwell †

ABSTRACT

This Article denies the sharply diametric rhetoric characterizing the debate over the viability of open source software. Rejecting the extremes, Satchwell argues that a middle solution exists whereby a properly formed intellectual property regime will incentivize the production of quality open source software while ensuring the stability of the open source paradigm and without disrupting more traditional intellectual property structures. To this end, this Article discusses the necessary focus and form of such an open source IP regime and shows that by concentrating exclusively on attribution and vertical stability a path may be steered between the tragedies of the commons and anticommons that will enable and promote open source as a viably sustainable production method. Finally, a possible form for such a system is suggested and its benefits discussed.

TABLE OF CONTENTS

---

* Winner of the 2005 Berkeley Technology Law Journal Student Notes & Comments Competition.

† B.A. 2003, Washington University; J.D. Candidate 2006, University of Pennsylvania. An earlier version of the paper was presented in Fall 2004 at the University of Pennsylvania. I am especially indebted to Professor Polk Wagner for his guidance, thoughts, and comments on early drafts of this piece. Comments appreciated: msatchwe@law.upenn.edu.

> The shape changes but not the form;
> The more it moves, the more it yields.
> More words count less.
> Hold fast to the center.
> —Chapter Five of the Tao Te Ching

I. INTRODUCTION: CAN'T WE ALL JUST GET ALONG?

The argument over open source software development has reached a near-fevered pitch. At one extreme are those who argue that open source software has no place in today's economy because without strong intellectual property controls, both the process and its products are doomed to failure.[1] At the other extreme are those who claim that while open source is the wave of the future, it can never operate productively under even weak intellectual property strictures.[2] As fervently as each of these positions is advocated, neither describes the actual current state of open source development. This Article will explain that between these two extremes lies a cohesive solution whereby open source production does not just coexist with a non-traditional intellectual property structure, but is in fact improved by intellectual property protections.

---

1. *See* Joseph Scott Miller, *Allchin's Folly: Exploding Some Myths About Open Source Software*, 20 CARDOZO ARTS & ENT. L.J. 491, 491-93 (2002) (quoting senior Microsoft executives who characterize "[o]pen source [as] an intellectual-property destroyer" and argue that "Linux is a cancer"); *see also* Mary Jo Foley, *Microsoft Adds to Its Anti-Open-Source Arsenal*, MICROSOFT WATCH, Sept. 13, 2004, http://www.microsoft-watch.com/article2/0,2180,1645122,00.asp; Jonathan Krim, *The Quiet War Over Open-Source*, WASH. POST, Aug. 21, 2003, at E1.

2. *See* Richard Stallman, *The GNU Operating System and the Free Software Movement*, *in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION (Chris DiBona et al. eds., 1999); Severine Dusollier, *Open Source and Copyleft: Authorship Reconsidered?*, 26 COLUM.-VLA J.L. & ARTS 281, 286-88 (2003).

While open source efforts have often resulted in viable (some might even argue superior) products,[3] it is important to query whether traditional intellectual property incentive structures are applicable. One major problem is that properly incentivizing individuals to participate in open source efforts has rarely resulted in financial remuneration. Although many current contributors appear to participate in open source development for little more than the colloquial prestige they stand to gain for their polished efforts, it is unclear whether such intangible benefits are a sufficiently effective motivation to drive large-scale efforts capable of rivaling traditional proprietary production. Further, it is empirically unclear whether intellectual property rights act as a motivation or as a limitation on those considering participating in open source development. Packaged with many open source projects currently distributed is some form of user license describing the rights and limitations of use allocated to the public. This Article attempts, on the most basic level, to assess how intellectual property rights can encourage production of viable, high-quality software developed through open source means.

To this end, two fundamental theories are presented. First, it must be determined whether or not intellectual property protections, in any form, make sense in the instance of open source development. Put another way, are there appreciable gains to be had by providing some form of proprietary control over what is fundamentally a reaction against traditional proprietary production schemes? I argue that some form of intellectual property rights would not only incentivize open source production but also increase the quality of the software produced through open source software development.

The second and more difficult question is what form this proposed intellectual property protection should take. In addressing this question, I will identify some clear shortcomings of traditional intellectual property schemes with regard to open source development. Employing basic economic analysis, I'll suggest a more optimal intellectual property scheme that is not only capable of properly incentivizing open source development, but also increasing product quality.

The problem is classic and well known. Ownership is, in effect, a right of exclusion. To possess property is fundamentally the ability to prevent others from using it. This rationale falls short, however, in terms of intel-

3. *See infra* notes 6-11.

lectual property. By its very nature, the value of such property is the innovation it represents, either in terms of a new expression of an idea or a new utilization of technology. Such innovations must be made manifest to be realized, but physical realization makes the innovation plain to others and may invite easy misappropriation.

Ownership with regard to intellectual property, therefore, has been implemented as a right of distribution more than as a right of exclusion.[4] In return for making her innovation known, the author/inventor is rewarded with the grant of some form of control over the distribution of manifestations of her innovation. In the Western legal tradition, this distribution control is provided primarily through patents and copyrights. This notion works well in the 'one creator, many users' paradigm classically preserved in traditional patent and copyright schemes. In fact, much of the structure of modern Western patent and copyright law revolves around the ways and means of preserving, protecting, and thereby rewarding only the distribution of the unaltered and unimproved form of a creator's work.[5] What is unclear is how well these traditional forms of intellectual property protection balance societal edification with innovator rewards in instances of non-traditional methods of production.

Of particular interest are recent efforts in open source development, especially those in the technology and software industry. In contrast with traditional proprietary or corporate production models, open source development is identifiable by two characteristics. First, open source projects are distributed in forms that allow not just observation and utilization but modification as well. In the software example, a particular program would be distributed in such a way that its human-readable source code is accessible in addition to its machine-readable object code. Second, open source projects are provided to users with a varying level of redistribution and modification rights. These rights range from a floor of very limited redistribution allowances accompanied by strict crediting requirements to a ceiling of unlimited modification, appropriation, assignment, and even resale.[6]

---

4. *See generally* Paul Goldstein, *Derivative Rights and Derivative Works in Copyright*, 30 J. COPYRIGHT SOC'Y 209 (1983) (discussing the evolution of American copyright law and examining rights in derivative works as distinguished from reproduction rights).

5. *See id.* at 216.

6. *See* Open Source Initiative, The Approved Licenses, http://opensource.org/licenses (last visited Nov. 13, 2005) (providing specific examples of redistribution rights). Also, bear in mind that open source software, while often billed as free software, is not necessarily gratis. This is an important distinction. While the terms "free software"

As such, famous open source efforts as Mozilla,[7] Apache,[8] and of course Linux[9] have shown open source development is becoming a major force in the software industry.[10] The idea is that by releasing to the public not just a piece of software itself, but human-accessible source code as well, a community effort may be used to develop and better the software in an often disorganized and highly egalitarian fashion as many individual programmers work on the software with minimal oversight. Instead of following the paradigm of traditional proprietary businesses where projects are developed in a classic factory-like hierarchy and traditional ownership notions are preserved, open source developers feel that many heads each working in its own direction are better than one tightly organized team.

In his article on the economic factors that motivate and sustain viable open source development, Yochai Benkler nicely synopsizes the basic theory of why open source development has become a practical alternative to more traditional proprietary production schemes:

---

and "open source software" are often used interchangeably in everyday conversation, some members of the open source community take such semantic issues very seriously. For the purposes of this Article such technical issues will be overlooked. The important point is that the "free" in free redistribution means the legal freedom to redistribute code, whether it is sold or given away without compensation.

7. Mozilla is an open source software suite including a web browser, e-mail utility, and other applications. Mozilla was spun off of Netscape as a non-profit open source effort in 2003. Firefox, the dedicated browser utility developed by the Mozilla team, has generated significant praise as a legitimate competitor to the Internet Explorer web browser developed by Microsoft. *See* CNET Editor's Review, http://reviews.cnet.com/Mozilla_Firefox_10/4505-9241_7-31117280.html; *see also* Cynthia L. Webb, *Firefox Flames Internet Explorer*, WASHTINGTONPOST.COM, Nov. 15, 2004, http://www.washingtonpost.com/wp-dyn/articles/A50983-2004Nov15.html.

8. Apache is an open source HTTP web server for UNIX platforms. As of August, 2004, approximately 67% of the web servers on the internet were running Apache software. Developed in early 1995, it is rapidly becoming the industry standard. *See* Apache Software Foundation, WIKIPEDIA: THE FREE ENCYCLOPEDIA, http://en.wikipedia.org/wiki/Apache_server (last visited Dec. 6, 2005).

9. Perhaps the best-known open source software in the world, Linux is a UNIX-like operating system kernel developed by Linus Torvalds and first released in 1991. Since then, the Linux kernel has been extensively and continuously overhauled by a world wide network of open source developers. The Linux kernel is used in many different Linux operating systems (distributed for free and for cost) and is running on twenty million computer systems worldwide.

10. *See* Arik Hesseldahl, *Better Browser Now The Best*, FORBES.COM, Sept. 29, 2004, http://www.forbes.com/2004/09/29/cx_ah_0929tentech.html?partner=tentech_newsletter (arguing that the open source Firefox browser is superior to its competitors). *See generally* News Forge, http://www.newsforge.com (an online newspaper covering Linux and open source developments around the world) (last visited Dec. 6, 2005)..

> Peer production [referred to here as open source development] has an advantage over firms and markets because it allows larger groups of individuals to scour larger groups of resources in search of materials, projects, collaborations, and combinations than is possible for firms or individuals who function in markets. Transaction costs associated with property and contract limit the access of people to each other, to resources, and to projects when production is organized on a market or firm model, but not when it is organized on a peer production model. Because fit of people to projects and to each other is variable, there are increasing returns to the scale of the number of people, resources, and projects capable of being combined.[11]

While open source efforts such as Mozilla and Linux have been held up as evidence that the model works, they have succeeded largely by contravening or even ignoring traditional intellectual property regimes. In other words, those open source efforts most often held up as examples of how peer production can prosper in the modern information economy have succeeded in spite of, rather than because of, contemporary intellectual property controls.[12] The existence of what are often self-serving defenses of the open source model are thus a deliberate condemnation of the narrow-mindedness of our current intellectual property structures. They are also implicit admissions that open source development still has a long way to go before it becomes more than a practical alternative and takes its place as a legitimate mainstream production option.

What follows is a discussion of how intellectual property rights, properly deployed, can improve and legitimize contemporary open source development. Part I provides the theoretical framework from which I will critique current open source IP rights systems and suggest my own solution. In Part II, I argue that open source development will benefit greatly from an IP rights system that serves the dual goals of incentivizing production and increasing the quality of products generated by focusing closely on proper attribution and vertical sustainability. Part III examines some well-known open source efforts that have succeeded by utilizing mechanisms very similar to the theoretical one outlined in Part I and II. These examples, nevertheless, serve as instances of sound theory, but imperfect implementation.

---

11. Yochai Benkler, *Coase's Penguin, or Linux and the Nature of the Firm*, 112 YALE L.J. 369, 376-77 (2002).

12. As will be discussed in Part IV of this Article, the use of open source software is premised on license terms that alter the author's conventional copyrights. As contracts between the author and the user delineating terms of use, these licenses are both inefficient and ineffectual.

From there I will move to Part IV, where I will suggest in more concrete terms a possible adjustment to our own IP system that would benefit society by improving and encouraging open source development as a more cost-effective and productive means of software development. Although not a magic pill to cure the open source debate, my solution will attempt to show that it is at least possible to productively integrate open source and intellectual property.

The argument presented does not attempt to define, nor provide, a new or in-depth explanation for the open source development model.[13] Rather, the model is taken as an established production approach that contrasts with traditional hierarchical and proprietary production schemes, and seeks to identify the shortcomings in traditional intellectual property regimes with respect to open source development and discuss some possible solutions.

II. HOW INTELLECTUAL PROPERTY RIGHTS CAN BENEFIT OPEN SOURCE DEVELOPMENT

Before addressing the obvious and significant question of what form an intellectual property regime capable of productive synchronization with open source development might take,[14] it is important to first ask what gains, if any, might be realized by attempting to marry intellectual property and open source development. In the open source development context, intellectual property rights in any form must decrease transaction costs. This can be accomplished by building an IP rights structure that focuses closely on serving two fundamental purposes: attribution and vertical sustainability.[15]

Much has been made of the argument that intellectual property is inherently incompatible with true open source development.[16] This argument is essentially that of the now famous tragedy of the anticommons, a situation held to occur when too many actors in a community hold rights

---

13. For an in-depth description of the open source model, see Marcus Maher, *Open Source Software: The Success of an Alternative Intellectual Property Incentive Paradigm*, 10 FORDHAM INTELL. PROP. MEDIA & ENT. L.J. 619 (2000). For a discussion of the fundamental economic and social motivations behind the open source model, see generally Benkler, *supra* note 11.

14. *See infra* Parts III and IV.

15. For the purposes of this article, "attribution" refers to accurate and sustainable notification authorship. Likewise, "vertical sustainability" describes a set of stable circumstances in which the rights regime adopted by an author is preserved as that original author's work is passed from user to user as well as to subsequent contributors.

16. *See supra* note 2.

of exclusion, such as property rights. This argument, first posited in 1998 by Michael Heller, contends that when such a situation occurs, many actors may collectively underutilize a resource even though they act separately and rationally.[17] The idea is that when an increasing number of actors are each assigned a property-like right, the effort required to reach a consensus amongst all who have a say in a matter will become greater and greater. Eventually the difficulty involved in getting everyone on board will become so high that some opportunities will simply go underutilized and some resources untapped. This is to say, in the end, that the tragedy of the anticommons is one of increasing transaction costs.[18]

Critics fear that liberal assignment of intellectual property rights in an information economy will result in precisely this sort of drastic increase in transaction costs. Because the open source model thrives fundamentally on the free exchange of information, these critics argue that even an incremental increase in transaction costs above absolutely minimal levels will smother any open source effort.[19]

Although the anticommons argument certainly has merits, it is important to recognize that the tragedy it foretells is not a necessary component of all intellectual property schemes. Increased fragmentation of exclusion rights does not necessarily imply higher transaction costs. Rather, transaction costs rise with the costs of knowledge and negotiation. This is one reason that the tragedy of the anticommons receives so much attention in intellectual property literature. Put another way, transaction costs rise in the anticommons model because it becomes difficult for any individual actor to: (1) ascertain who the pertinent rightholders are with regards to a project that actor wants to complete, and (2) convince them to deal with each other usefully and efficiently.

In the intellectual property realm, it may not be obvious who controls the rights to basic technologies or information necessary to the completion of larger projects; nor is it always easy to entice those holding the rights to act in strategic concert. The more a creative field is fragmented by the as-

---

17. Michael A. Heller, *The Tragedy of the Anticommons: Property in the Transition from Marx to Markets*, 111 HARV. L. REV. 621 (1998).

18. Anticommons property is prone to the tragedy of underuse. Once anticommons property appears, neither markets nor subsequent regulation will reliably convert it into useful private property, even if the property rights are 'clearly defined' and contracts are subject to the 'rule of law.' Transaction costs, holdouts, and rent-seeking may prevent economically justified conversion from taking place.

Id. at 687-88.

19. *See generally* Ruth L. Okediji, *Trading Posts in Cyberspace: Information Markets and the Construction of Proprietary Rights*, 44 B.C. L. REV. 545 (2003).

signment of rights to holders of smaller and smaller parcels of information, the more actors must be identified and involved in any cumulative endeavor and the more strategic action issues come into play. In short, because creativity is fundamentally an act of consolidating ideas,[20] the risk of killing it through the overassignment of intellectual property rights is salient. With this tragic scenario in mind, it becomes clear that minimizing transaction costs must be a central aim of any intellectual property regime that seeks to accept, let alone benefit, open source development.

A. The Importance of Attribution and Sustainability

Creating a rights allocation system for open source development that minimizes transaction costs is not only theoretically possible, but practically feasible without resorting to a commons-based system. In order to avoid the type of spiraling transaction costs associated with rights fragmentation, any hypothetical intellectual property system must promote both transparency and avoid nonproductive and inefficient negotiations between actors.

Proper attribution is one way a system of allocating and implementing intellectual property rights might deal with the problem of increased transaction costs associated with the difficulties inherent in figuring out which individuals hold rights to material necessary to software development. One significant contemporary barrier to open source development as a legitimately competitive software production model is the difficulty any individual developer has in ascertaining who she needs to ask permission of to use code she may want to implement into her project.[21] In the analog world, if I want to insert an excerpt from an article written by someone else into a book I am writing, I need only to look to the article's author and perhaps the journal in which it was published to ask for permission to use the excerpt. Because the article's content is clearly attributed to the

---

20. "Masterpieces are not single and solitary births; they are the outcome of many years of thinking in common, of thinking by the body of the people, so that the experience of the mass is behind the single voice." VIRGINIA WOOLF, A ROOM OF ONE'S OWN (1929); *see also* William M. Landes & Richard A. Posner, *An Economic Analysis of Copyright Law*, 18 J. LEGAL STUD. 325, 332 (1989); R. Polk Wagner, *Information Wants to be Free: Intellectual Property and the Mythologies of Control*, 103 COLUM. L. REV. 995, 998 (2003).

21. For a discussion of this phenomenon that occurs when too many fragmented exclusionary rights are granted as it pertains in the realm of patents, see Rosemarie Ham Ziedonis, *Don't Fence Me In: Fragmented Markets for Technology and the Patent Acquisition Strategies of Firms*, 50 MGMT. SCI. 804, 807-08 (2003). Although this article deals primarily with the anticommons issues connected with patents, the same transaction cost situations and arguments apply equally well in terms of copyrights.

author and its distribution to the publisher, it is a trivially simple task for me to secure the permission necessary to use that author's words in my own work. Of course, I too must accurately attribute the portion of work that is not mine so that any subsequent authors who might want to excerpt me do not inadvertently take something from the author I excerpted without asking. The point is that because conventional copyright ensures that everyone knows who is responsible for what work, the transaction costs involved in securing proper permissions is minimal.

This is not the case in the software development industry. In order for a computer program to be written and manipulated by a human as well as understood and utilized by a computer, any given piece of software must be expressed in two forms that look different but are functionally equivalent: object code and source code. Source code is human-readable and in a form that software coders can easily interact with. Object code is machine-readable; it is entirely ones and zeroes—the program reduced to pure binary format.[22]

Open source development efforts often run into trouble in terms of prohibitively high transaction costs because of a conspicuous lack of access and attribution at the source code level. Any IP rights system that seeks to improve the viability of open source development must focus on this area because it is code that can be understood and manipulated by people. Because software is often at least partly a compilation of tried-and-true code, software engineering is rarely done from scratch.[23] Proper attribution therefore plays an essential role in a successful open source intellectual property system by ensuring that not only would the software authors receive credit (and any available compensation) for their work, but

---

22. The major distinguishing difference between open source software development and proprietary efforts is that while both the source code and object code for open source software is released, corporations like Microsoft only release the object code for their software. Further, the use of the object code for such programs is very often contingent upon the user accepting license terms that forbid attempting to alter or even access the underlying source code. Take for example this license language from Microsoft's XP Professional operating system:

> 5. LIMITATION ON REVERSE ENGINEERING, DECOMPILATION, AND DISASSEMBLY. You may not reverse engineer, decompile, or disassemble the Product, except and only to the extent that it is expressly permitted by applicable law notwithstanding this limitation.

Microsoft, Microsoft Windows XP Professional END-USER LICENSE AGREEMENT, http://www.microsoft.com/windowsxp/pro/eula.mspx (last visited Dec. 6, 2005).

23. *See* Maher, *supra* note 13, at 624-25 (discussing the widespread practice of reusing old code in new projects).

also crucially reducing transaction costs by allowing those who wish to use code to identify those with whom they must negotiate.

However, identifying those who hold copyrights in code that a software engineer would like to use in a new program is only half the battle. The other factor that significantly increases transaction costs is the effort required to get those rights-holders to negotiate and agree to provide the hopeful engineer with the permission to use their work. Therefore, in addition to mechanisms for proper and pervasive attribution, a successful open source intellectual property system must also be vertically sustainable. What good would it do, in other words, for a given pool of collaborators to work hard on an open source project only to have some outside individual swoop in and unilaterally appropriate and commercialize the project as soon as it reaches fruition? What about the corporate software producer who would like to realize the efficiency and quality gains to be had by adopting open source production methods but does not want her competitor to simply walk off with the results?[24]

To avoid such unilateral downstream appropriation of the fruits of open source labors, the assignment of IP rights must strike a balance between providing authors with sufficient control over their work to encourage them that it is worthwhile to produce while at the same time ensuring that the information and otherwise protected material produced remains 'free' enough to others that downstream production may progress unfettered. By achieving such a vertically sustainable balance a proper open source IP system would drastically decrease, if not eradicate, the transaction costs associated with motivating rightholders to collectively act. Further, proper vertical sustainability would ensure that no individual rightholder could prevent downstream development by refusing to share her work.

III. THE EFFECTS OF ATTRIBUTION AND VERTICAL SUSTAINABILITY ON OPEN SOURCE DEVELOPMENT

If done properly, any system designed to allocate and protect intellectual property rights would serve the dual goals of incentivizing individuals to produce while at the same time increasing the quality of their work. Put another way, society wants not only more software, but software that works better. If a system of intellectual property is to not just coexist with, but actively improve open source development, it must encourage and improve production.

---

24. *See* Benkler, *supra* note 11, at 439-40 (discussing the de-incentivizing results of unilateral downstream appropriation).

In order for open source development to compete with traditional proprietary assembly as a legitimate means of production in general, and as a viable means of software development in particular, open source production must be incentivized.[25] At one extreme, to simply abolish property rights in information and hope to incentivize production through a complete rejection of proprietary rights would result in either far less work getting done, or a tragedy of the commons-like race to the bottom in terms of quality of work produced.[26] On the other hand, the opposite extreme of strict intellectual property controls quickly results in the type of spiraling transaction costs described by the tragedy of the anticommons that precludes any effort at open source development.

If our goal is to harness the utility, efficiency, and innovative solutions of open source then we must choose the middle path: without completely rejecting or embracing traditional intellectual property notions we must implement a system of IP rights that creates sufficient incentives for open source production while at the same time minimizes transaction costs for all actors. I agree with Benkler's argument that "[a]t the broadest level, wherever peer production can motivate behavior better than markets or firms, then certainly it will be superior."[27] An IP system that properly focuses on attribution and vertical sustainability is a powerful way to increase the ability of open source to motivate behavior in addition to improving the quality of product.

One of the most common arguments raised against synchronizing intellectual property with open source development is that assigning rights will stifle production.[28] Indeed, the open source movement itself is largely conceptualized as a reaction against the overbearing control embodied by

---

25. Indeed, because of the uncertainty of monetary returns on open source participation relative to proprietary production, incentivization has been identified as perhaps the largest hurdle to enabling open source production as a legitimate market force. For a discussion of these issues in the open source software development context, see Benkler, *supra* note 11, at 423-34. For a more fundamental treatment, see Goldstein, *supra* note 4, at 216-17.

26. For a discussion on the failure of the abolition of IP rights, see generally Wagner, *supra* note 20.

27. Benkler, *supra* note 11, at 426.

28. The preamble of the GNU General Public License articulates this fear:

> Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

GNU General Public License, Version 2, Free Software Foundation, Inc. (1991), http://www.fsf.org/licensing/licenses/gpl.html.

perfect intellectual property rights.[29] While I certainly agree that traditional IP structures are at best irrelevant and at worst destructive to open source development, I strongly disagree that open source is necessarily IP averse.

The proper attribution and sustainability that may be achieved through IP rights go a long way towards providing the kinds of incentives to production that monetary compensation usually performs in more traditional publishing or production industries. By providing pervasive attribution throughout a vertically sustainable chain of open source development, the identity of those individual contributors most responsible for the success of some collaborative effort will be known to all other participants in the project, as well as any eventual consumer of the project. Indeed, it is exactly this kind of notoriety that many current open source developers seek.[30]

While it is unclear whether these developers covet this notoriety for purely hedonistic personal reasons or because it occasionally may be parlayed into more economically rewarding employment,[31] the proper attribution available through a system of IP rights would serve both aims. By ensuring that an author's name will not be disassociated from her work, no matter who utilizes the software or how the software is put to use, the author is reassured that her peers and consumer of the software will credit her for producing the viable software.

By properly attributing an author's work as a matter of law and vertically sustaining that attribution, many authors who might otherwise not contribute to open source efforts could be enticed to do so because they have less reason to think that their work will go unnoticed. They are also more likely to capture the colloquial or economic prizes that come with notoriety. In other words, it is possible to replace the incentives provided by money (available only to the few)[32] with those provided by property

---

29. *See* Lawrence Lessig, *The Architecture of Innovation*, 51 DUKE L.J. 1783, 1788 (2002) (discussing the motivations of the open source movement as they relate to IP structures).

30. *See* Maher, *supra* note 13, at 631-36 (discussing the high value many open source developers place on visibility within the community); Benkler, *supra* note 11, at 423-26 (discussing the open source motivations).

31. *Id.*

32. Even though the majority of software used today for everyday home computing tasks is proprietary, the number of individuals financially compensated for this proprietary production is very small relative to consumer base of the software they produce. As any computer hobbyist or amateur hacker would probably tell you, it's hard to get paid to work with code. Firm statistics on how many people are being paid to code in America are difficult to come by, as are statistics counting how many pieces of software are sold

rights (available to all),[33] while at the same time increasing access to traditional corporate rewards for production.[34]

Additionally, an IP regime focused on attribution and vertical sustainability will further incentivize production by lowering barriers to entry such that many whose perceived benefit may nevertheless be marginal will still be willing to participate usefully in the project.[35] As open source development now stands, many potential contributors do not participate because of the real and perceived barriers to entry that anticommons-type transaction costs represent. Decreasing those transaction costs through: (1) promoting transparency and knowledge of rightholders through proper attribution, and (2) crafting a vertically sustainable rights balance that minimizes the costs of negotiation and the ability of the few to derail the

---

each year. That said, a rough calculation is possible. In May 2003, the U.S. Department of Labor's Bureau of Labor Statistics reported that there were a little over one million Americans working as computer programmers and software engineers. U.S. DEP'T OF LABOR, BUREAU OF LABOR STATISTICS, 2003 National Employment and Wage Data from the Occupational Employment Statistics Survey by Occupation, http://www.bls.gov/oes/2003/may/table1.pdf. The number of Americans writing software thus pales significantly when compared to the 32 million copies of its Windows XP operating system that Microsoft sold in the first six months it was available. Margie Semilof, *XP Sales Exceed 32 Million; SP1 Set for September*, SEARCHWIN2000.COM, June 28, 2002, http://searchwin2000.techtarget.com/originalContent/0,289142,sid1_gci836066,00.html. Put another way, in just six months one company sold at over 30 times as many copies of just one computer program as the number of paid programmers and software engineers in the entire U.S.

33. By their fungible nature, property rights are literally available to all. This is significant because these fungible goods are freely given (in terms of copyright) and can be secured regardless of nationality or citizenship.

34. One of the most important aspects of the proper attribution afforded by a well-considered IP rights regime is the notoriety available to clever programmers. Indeed, the informal recognition afforded the best open source contributors can result in gainful employment and/or project contracts. For one perspective, see Open Source, Jobs for Hackers, http://opensource.org/advocacy/jobs.php (last visited Dec. 6, 2005). For an accessible and in-depth discussion of the incentive structures inherent to open source, see Josh Lerner & Jean Tirole, *Some Simple Economics of Open Source*, 50 J. INDUS. ECON. 197, 212-20 (2002).

35. Benkler speaks at length about several such successful open source efforts that employ low barriers to entry. *See* Benkler, *supra* note 11, at 384-89. Benkler attributes much of this barrier lowering function to the granularity of the projects he discusses. Although granularity is an important aspect of open source development, it is by no means fundamental to successful open source efforts. With regard to lowering barriers to participation, decreasing transaction costs across all fronts through proper IP is the crucial prerequisite. Once that function is performed, granularity will often follow on its own accord.

work of the many, will mean that even individuals who perceive only minimal benefits of participation will nonetheless be incentivized to do so.

By rewarding open source producers with attribution (leading to sustainable property rights and minimizing the transaction costs that bar entry to a level below even marginal rewards for participation), a proper IP rights framework can seed the open source model with sufficient incentives to promote greater production. Indeed, rather than stifling creativity and invention, a proper IP framework with regards to open source production can in fact result in more and better products.

A. Incentivizing Open Source Production

One of the most often repeated arguments against the long-term sustainability of open source development revolves around the problem of enticing individuals to participate in open source efforts.[36] According to this argument, while there may be an extremely large user base for popular software, enticing those users to become active contributors rather than just passive customers is difficult, and they are likely to do so only temporarily.[37]

While conventional wisdom holds that it's nearly impossible to get someone to do something for nothing, there are a variety of non-monetary indirect appropriation means available to open source developers, in addition to the significant economic rewards reaped by the authors of successful software. In his article *Coase's Penguin, or, Linux and the Nature of the Firm*, Benkler expands on the work of Lerner and Tirole to abstract the mechanics of human motivation.[38] Benkler's framework is both useful and accessible, and I will use it here to motivate my own argument.

Benkler identifies three separate rewards that may motivate behavior: (1) monetary rewards, (2) hedonistic rewards associated with the subjective personal pleasure derived by each agent, and (3) social-psychological rewards tied to the social associations and status perceptions tied to the activity in question. He then identifies a rewards function:[39]

$$R=M+H+SP$$

---

36. *See supra* note 25.

37. *See* Maher, *supra* note 13, at 628-37 (discussing the process of transforming code users into code contributors).

38. *See supra* note 11.

39. Benkler's argument and mathematical abstractions are in fact quite a bit more subtle and complex. The simplified version of Benkler's arguments I present here preserves the substance of his ideas while providing a sufficiently basic framework upon which to base my own thoughts.

According to Benkler, rational actors will base their decisions on whether to act or not on the value of *R*, not on *M*, *H*, or *SP* alone. Given this framework, it is important to ask what role intellectual property rights play in the subjective valuation of *R*. While Benkler does argue that traditional, "strong" intellectual property rights will stifle open source development, he says nothing of the non-traditional, moderate IP rights focused on attribution and vertical sustainability that I advocate.[40]

Clearly, proper attribution can have an immensely positive effect on *SP*. In any form, the availability of social-psychologic rewards is closely tied to the actor being recognized for his contribution. By assigning vertically sustainable IP rights that preserve an author's association with his work even through successive generations of use, reinterpretation, and reuse, many individuals' subjective assessment of *SP* will greatly increase. In this way code that is particularly clever, adaptable, or otherwise useful will be properly credited to its author and that author will reap the benefits in positive *SP* rewards. Paying credit where credit is due is one of the quickest ways for informal rewards like prestige and professional reputation to begin directly incentivizing open source development.[41]

A potential open source contributor's highly subjective valuation of the hedonistic rewards associated with a project can also be strongly influenced by an IP rights regime that focuses on attribution and vertical sustainability. *H* rewards can be conceptualized as the individual personal joy a contributor derives from participating in an open source project. While there is evidence that few open source contributors participate out of pure altruism, human nature dictates that most derive more pleasure from participating in projects they consider worthwhile. Two conjectures can thus be drawn. First, the majority of open source contributors who are not con-

---

40. Benkler argues that:

> Strong intellectual property rights, in particular rights to control creative utilization of existing information, harm peer production by raising the cost of access to existing information resources as input. This barrier limits the capacity of the hundreds of thousands of potential contributors to consider what could be done with a given input and to apply themselves to it without violating the rights of the owner of the information input. This does not mean that intellectual property rights are entirely bad. But we have known for decades that intellectual property entails systematic inefficiencies as a solution to the problem of private provisioning of the public good called information. The emergence of commons-based peer production adds a new source of inefficiency.

Benkler, *supra* note 11, at 445-46. It is exactly these types of systemic inefficiencies inherent in the current IP model that the moderate solution I propose in Part IV seeks to overcome.

41. *See supra* note 34.

tributing for purely altruistic reasons would enjoy participating even more if they knew their contributions would be attributed to them. Second, because most contributors are accepting the opportunity cost of participating in an open source project rather than a generally higher paying proprietary effort, it is unlikely that they would like to see their work appropriated for someone else's exclusive gain.[42] Therefore more vertically sustainable open source projects are likely to be considered more worthwhile.

Considering this capacity for attribution and vertical sustainability to positively influence both *SP* and *H*, it is possible to adjust Benkler's basic equation to reflect the influence of intellectual property rights on *R*:

$$R=M+(H+SP)(IP+1)$$

In this new equation, the variable *IP* symbolizes the effects of attribution and vertical sustainability as applied to either *H* or *SP*. When the effects of intellectual property are either absent or ignored, this equation reduces to Benkler's basic assertion of motivation. However, when the positive effects of attribution and vertical sustainability available through a proper intellectual property arrangement are considered, *R* will increase and open source production will be incentivized.[43]

Examining this equation reveals that whenever the effects of intellectual property can be realized positively, it is possible to overcome small immediate monetary returns with enhanced social-psychological and hedonistic rewards. This is particularly significant with regards to incentivizing and sustaining open source production. Benkler's analysis prompts him to conclude that there are "a series of likely conditions under which nonproprietary organizational approaches will be sustainable. First, there

---

42. Unilateral downstream appropriation is a major issue in the open source community. Richard Stallman has argued that the kind of copyleft licenses discussed in Part III of this Article is the only way to avoid such unfair appropriation. *See* Why Copyleft?, http://www.gnu.org/philosophy/why-copyleft.html (last visited Nov. 13, 2005); *see also* Maher, *supra* note 13, at 677-78 (discussing unilateral appropriation).

43. If *(H+SP)* were multiplied by only a factor of *(IP)* then the equation would reduce to *R=M* if zero IP benefits were realized. This is clearly not the case; even if no IP benefits exist there can remain very real *H* and *SP* rewards. Because proper IP rights increase *R* whenever they are realized, *H* and *SP* must be multiplied by *(IP +1)* to maintain them even if IP benefits are absent. Additionally, the equation, as written, implies that *IP* operates equally on *H* and *SP* in a given situation. This may or may not be true. As discussed above, *IP* effects that promote vertical sustainability, for example, may have a greater effect on *H* than on *SP*. That said, this minor technical inaccuracy does not decrease the conceptual accuracy and is acceptable in order to maintain the simplicity of the equation.

is the case of projects . . . where market remuneration would likely be too costly to sustain, but where hedonic and social-psychological rewards can provide contributors with positive rewards."[44] In other words, where *(H+SP)>>M, R* may still be sufficiently large to prompt participation in open source efforts. Because IP acts directly on *(H+SP)* in a positive fashion, an intellectual property rights regime focused on attribution and vertical sustainability will reinforce the commitment made by those who would have been enticed to participate without any IP rights, and more importantly, incentivize more people to contribute who otherwise would not have done so due to insufficient *(H+SP)* valuation.

Benkler continues his observation: "[s]econd, there are instances where the value of monetary return is small relative to the value of the hedonic and social-psychological rewards, particularly where the cultural construction of the social-psychological rewards places a high negative value on the direct association of monetary rewards with the activities."[45] In these instances, *(H+SP)* is again much greater than *M*. The difference is that in the first set of projects, the inequality is driven by objective market forces whereas in these second instances, the inequality is driven by social context and subjective valuations. Nevertheless, the beneficial operation of intellectual property rights remains the same.

B. Maximizing the Quality of Open Source Products

After facilitating the delivery of more products to the public, the second goal of any useful IP rights system should be improving the quality of the products delivered. This Section will explain how intellectual property rights can not only incentivize production, but also increase the quality of the work produced.

As with incentivizing production, improving the quality of products through intellectual property rights is a function of proper attribution and vertical sustainability. One first order result of lowering transactional costs to their absolute minimum is that any interested party may easily participate in an open source development effort.[46] It stands to reason that those who choose to participate are self-selected as the most personally interested individuals available. In contrast to traditional corporate production where each rational producer only works as hard as she must to ensure her continued employment, the fundamentally decentralized open source model encourages voluntary participation by those who most wish to do

44. Benkler, *supra* note 11, at 433.
45. *Id.* at 434.
46. *See supra* note 35.

so.[47] All things being equal, those who desire to perform a task will do a better job of it than those who are required to do the task. Therefore, the eradication of barriers to entry represented by an IP system that properly attributes and preserves vertical sustainability will result in more self-motivated people generating higher-quality work.

Because the open source development model is decentralized and largely non-hierarchical, there is rarely a finite and clearly defined goal, or a point at which a project is considered complete.[48] Rather, open source development may be considered more of a rolling process where improving on the work of those who came before is often the most tangible goal. For instance, compare the incrementally improved and highly frequent releases of software, such as the Fedora Project distribution of the Linux operating system, to the universally overhauled releases of a new Microsoft operating system that occur only once every few years.[49] This is largely a result of the fact that open source is by definition open, permitting and encouraging each interested user to immediately improve on the software she receives. In contrast, software developed by traditional corporate means is

---

47. *See* Maher, *supra* note 13, at 631-37 (discussing the open source culture); Benkler, *supra* note 11, at 372-73 (discussing the application of organization theory to the open source paradigm).

48. This open source characteristic is displayed in the Frequently Asked Questions section of the Fedora Project's (an open source Linux software suite) webpage:

> Q: Why a project instead of a product?
>
> A: A global steering committee at Red Hat decided that Red Hat Linux was suffering from too many compromises as a retail 'product,' and that we should redirect our efforts at creating a community-based project. Rather than being run through product management as something that has to appear on retail shelves on a certain date, Fedora Core will be released based on schedules, set by a steering committee, that will be open and accessible to the community, as well as influenced by the community.

The Fedora Project FAQ, http://fedora.redhat.com/about/faq (last visited Oct. 13, 2005).

49. For instance: Microsoft Windows was first announced in 1983. Windows 1.0 was released in 1985, 2.0 in 1987, 3.0 in 1990, Windows 95 was released in 1995, Windows 98 was released in 1998, and Microsoft Windows XP first appeared in 2001. That's an average of one release every 2.6 years. In stark contrast, the Fedora project (an open source Linux-based operating system) website announces updates to the Fedora Core code a few times a month, and makes incremental updates reflecting the work done each day available every night. *See* Fedora Project Schedule, http://fedora.redhat.com/participate /schedule (last visited Oct. 13, 2005) (laying out version release dates).

almost always only available on the condition that it is not tampered with, let alone altered.[50]

This means that there is a much higher availability of value-added software developed through open-source means than that churned out by hierarchical corporate production. For instance, while the basic Linux operating system is available for free, several value-added versions developed by open source means are sold. Further, as new uses and issues are identified for which the software might be adapted, a community of open source developers is constantly and instantly available to shape the software to meet these needs.[51]

This responsiveness to community needs is especially relevant in the world of software where the cost of producing good software is rarely different from the cost of producing bad software, but where the use of good software can save billions of dollars compared to the use of bad.[52] Funda-

---

50. To be sure, when viewed up close, decentralized production appears extremely inefficient. Whereas traditional proprietary production adheres to clearly defined goals and processes, the efforts of individual actors participating in a decentralized production effort (like open source development) are often redundant or at odds. When considered en masse, however, the efforts of each decentralized actor overlap and the group as a whole moves in the direction of general improvement. A visual metaphor might be a colony of ants scavenging for food. If you watch the behavior of each individual ant they seem extremely unorganized. And they are. Some ants find food, some ants wander aimlessly, and some ants switch back and forth. But when you consider the behavior of the colony as a whole, the massively paralleled distributed efforts of each individual ant combine to assure that the colony is fed. For a far more expert discussion of the considerable power of decentralized production, see Eric Steven Raymond, *How Many Eyeballs Tame Complexity*, *in* THE CATHEDRAL AND THE BAZAAR, http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar (last visited Oct. 25, 2005).

51. Indeed, it seems that the number of internet chat rooms and running bulletin board services devoted to open source grows daily. At the time of writing, the web search engine Google found over eleven million newsgroup postings with the word "linux" in their title, and over 2,000 individual newsgroups devoted to linux. *See* Google Groups, http://groups.google.com (last visited Nov. 21, 2005).

52. In one instance, a:

> security vulnerability in Microsoft's database program SQL Server 2000 enabled a computer worm known as Slammer to temporarily take down hundreds of thousands of computers around the world, causing an estimated $950 million to $1.2 billion in lost productivity worldwide. Among the reported damage, 13,000 automated teller machines temporarily stopped working, some airline flights were canceled, and the emergency 911 system in at least one city temporarily stopped working.

Reid Goldsborough, *Is There a Solution to Buggy Computers?*, OFFICESOLUTIONS, July-Aug. 2003, *available at* http://www.findarticles.com/p/articles/mi_m0FAU /is_4_20/ai_105367980. Jack Ganssle has commented that because much of the world's economy is directly tied to software, "fire code"-like rules should be installed to minimize the im-

mental institutional barriers often result in the first release of corporate-developed software being buggy or malfunctioning.[53] Because she is not allowed to tinker with the software she has purchased, the consumer must wait until the corporation fixes the problem (*if* the corporation fixes the problem).[54] On the other hand, the decentralized rolling nature of open source development means that if one proposed solution creates more problems than it solves, it can either be adjusted or discarded relatively immediately by the open source development community.[55]

On a superficial level, proper attribution's role in increasing *SP*-type rewards will encourage individuals to produce better products since few, if any, would want to be associated with inelegant or failed solutions. But there is a more fundamental aspect of open source that allows proper IP to truly maximize quality: open source software is designed to solve problems whereas proprietary software is designed to compete in the market.[56] Because the primary purpose of the software as conceptualized by its manufacturer is to compete in the marketplace, clever solutions to actual problems are only one of the many ways in which this goal might be realized. The result is the tendency for proprietary software to lean towards an emphasis of style over substance in comparison to open source software's often more elegant, if less accessible, solutions.[57] Contextualizing this disparity, one commentator has written:

> [I]n every release cycle Microsoft always listens to its *most ignorant customers*. This is the key to dumbing down each release

---

plementation of bad software and the significant monetary and human costs such software entails. Jack Ganssle, *Codifying Good Software Design*, Aug. 5, 2004, http://www.embedded.com/showArticle.jhtml?articleID=26806185.

53. Proprietary corporations almost always announce upcoming software releases in advance in order to build consumer and investment interest. Because these corporations stand to lose significant amounts of money if they miss their self-imposed software release dates, the corporations often release code before it is fully tested.

54. *See infra* note 57.

55. Indeed, this parallel peer review is one of the defining traits of open source development. *See* Maher, *supra* note 13, at 626-28.

56. This distinction is one of motivation, an *ex ante* reason the software is developed. Of course, both proprietary and open source software, once released, compete in the marketplace. It is perhaps then more accurate to say open source software is almost always modified by individuals seeking to solve a particular problem, while proprietary software is modified by corporations seeking to capture a larger market share, and sometimes seek to satisfy this goal by solving problems.

57. Again, because proprietary producers seek to capture larger market shares they often attempt to do so by improving the superficial design of software rather than focusing on its function. Open source, on the other hand, has routinely been criticized as difficult for the average user to access because of its attention to function over useability.

> cycle of software for further assaulting the non personal-computing population. [Open source developers for both] Linux and OS/2 developers, [on the other hand,] tend to listen to their *smartest* customers. . . . The good that Microsoft does in bringing computers to non-users is outdone by the curse they bring on experienced users, because their monopoly position tends to force everyone toward the lowest-common-denominator, not just the new users.[58]

These benefits of open source's incremental and solution-motivated approach to software development can be maximized by an IP rights system that preserves vertical sustainability because a system that allows users to not just utilize but adapt software to their own needs will by necessity be more responsive to community desires.

Further, vertical sustainability as it is conceptualized here does not preclude the possibility of economic gain.[59] There is no reason why an individual who develops a particularly useful or clever program having added her own work to a framework of open source code should be precluded from commercializing her product. In fact, one of the most salient benefits of the open source development model is its propensity for producing value-added software. Although open source does not mean monetarily free,[60] rational consumers will not pay for software developed through non-proprietary means unless it stands head and shoulders above

---

58. Tom Nadeau, Learning from Linux: OS/2 and the Halloween Memos, http://www.os2hq.com/archives/linmemo1.htm (last visited Oct. 23, 2005)

59. Indeed, many open source companies compete successfully with proprietary corporations. Robert Merges has argued that companies such as IBM can use open source production as an economic strategy in the marketplace:

> IBM's contribution to and backing of Linux comes free of property right claims. IBM's work product becomes part of the public domain. This both permits IBM to draw on the work of previous contributors, and (key for this argument) encourages downstream users to adopt Linux without the fear of being held hostage by IBM. IBM's investments are P[roperty] P[reempting] I[nvestment]s, precluding anyone (including IBM itself) from claiming property rights in the operating system. This credibly assures other firms that IBM will not assert the kind of control over the Linux operating system that other firms fear Microsoft will assert or has asserted over Windows.

Robert P. Merges, *A New Dynamism in the Public Domain*, 71 U. CHI. L. REV. 183, 193 (2004). Even Richard Stallman has written a piece on the marketability of open source software. *See* Selling Free Software, http://www.gnu.org/philosophy/selling.html (last visited Oct. 23, 2005).

60. Richard Stallman has commented that "To understand the concept, you should think of 'free' as in 'free speech,' not as in 'free beer.'" The Free Software Definition, http://www.gnu.org/philosophy/free-sw.html (last visited Oct. 25, 2005).

what they can get without paying. The incentive to add value to the bare code through service, documentation, support, or other tangential means is therefore very high.[61] This means that developers will benefit from the pool of knowledge and economic efficiencies inherent in vertically sustainable open source development, while at the same time be encouraged to produce quality software because of the financial rewards for value-addition they may realize through proper attribution. In this way, intellectual property rights, properly implemented, may provide ex ante incentives and facilitate ex post rewards for producing the most quality software possible.

The ability to produce high quality software is critical to the survival of open source development. IBM's recent and significant investment in non-proprietary production as well as Microsoft's attacks on open source signify that the open source community recognizes the importance of quality and has embraced this goal sufficiently enough to inspire the confidence (IBM) and ire (Microsoft) of two of the world's biggest tech companies.[62] Rather than trusting the market, sympathetic proprietary corporations, or the decentralized commitment of the open source community to preserve an ongoing campaign for quality, IP rights present a discrete and predictable way to realize and maximize the open source movement's natural and necessary appetite for effective and useful software.

C. Overcoming Barriers to Open Source Participation

Incentivizing open source production of quality software through assigning IP rights focused on attribution and vertical sustainability will do little good if those rights do not also help overcome barriers to rational participation in an open source effort. In addition to incentivizing participation, a proper IP regime can help overcome the two barriers to open

---

61. As the Open Source Initiative puts it on their website FAQ:

> [Q:] How do I make money on software if I can't sell my code?
>
> [A:] You can sell your code. Red Hat does it all the time. What you can't do is stop someone else from selling your code as well. That just says that you need to add extra value to your code, by offering service, or printed documentation, or a convenient medium, or a certification mark testifying to its quality.

Open Source Initiative, *Frequently Asked Questions*, http://opensource.org/advocacy/faq.php (last visited Oct. 23, 2005).

62. For a discussion of IBM's close ties to the open source community, see Cynthia L. Webb, *IBM's Open-Source Lovefest*, WASHINGTONPOST.COM, Sept. 13, 2004, http://www.washingtonpost.com/wp-dyn/articles/A17842-2004Sep13.html?nav=rss_technology. For a discussion of Microsoft's opposition to open source, see Ben Charny, *Microsoft Raps Open Source Approach*, CNETNEWS.COM, May 3, 2005, http://news.com.com/Microsoft+raps+open-source+approach/2100-1001_3-257001.html.

source mentioned most often: integration failures[63] and downstream appropriation.[64]

The problem of integrating the efforts of a large number of diverse participants is a significant issue in a deliberately decentralized and non-proprietary production model such as open source development. Without an effective integration strategy individual contributors' efforts will not be assembled into an aggregate project. This ex post failure to harness the creativity and effort of open source participants will immediately and severely decrease each contributor's *H* and *SP* valuations to the point that she will view her contribution as wasted effort. Clearly any contribution IP could make towards easing integration would be highly beneficial.

Again, by focusing on attribution and vertical sustainability, an IP system can facilitate easier project integration. Accurately preserved attribution facilitates the type of constructive peer review that helps choose which contributions should be included in the final project.[65] Attribution also associates the author's identity with a particular program or piece of code, implying that each contribution is more substantial personally than an anonymous submission. Most importantly, attribution provides the means though which both contributors and compilers can track each individual's modular progress and status throughout the integration process.[66]

Even if a project is successfully integrated, however, contributors will still feel that their work was for naught if some downstream actor is able to unilaterally appropriate the project and exclude the contributors from further participation and its associated rewards. The most common way in which this occurs is when a project is completed in an open source mode but the source code of the finished product is then removed from the commons and held as a proprietary secret.[67] Indeed, it is this fear of failed

---

63. *See* Lerner & Tirole, *supra* note 34, at 220 ("The success of an open source project is dependent on the ability to break the project into distinct components."). Maher also discusses the benefits of what he calls the "modularity" of open source projects. Maher, *supra* note 13, at 636-37.

64. *See* Benkler, *supra* note 11, at 439 ("The more important potential 'defection' from commons-based peer production is unilateral appropriation.").

65. This peer review structure is well-known in academia. For a discussion of peer production as it applies to scientific research, see Yochai Benkler, *Commons-Based Strategies and the Problem of Patents*, 305 SCI. 1110-11 (2004).

66. *Cf.* Lerner & Tirole, *supra* note 34, at 218-20 ("It is clear that giving credit to authors is essential in the open source movement.").

67. Stallman and many others argue that the only way to absolutely ensure vertical sustainability is through copyleft type licenses such as the GNU GPL. *See* Copyleft: Pragmatic Idealism, http://www.gnu.org/philosophy/pragmatic.html (last visited Nov. 5,

sustainability that prevents many erstwhile open source contributors from continuing their work.

The specific purpose of the vertical sustainability aspects of the IP regime suggested here is to prevent this type of downstream appropriation. By ensuring that downstream actors cannot benefit from open source developers' work without their permission the developers will be confident that although they release the substance of their work to the public, they can retain control over the strategic decisions of how that work is directed. Further, vertical sustainability requires that there be a sufficient volume of shared resources to sustain open source development over the long run. The goal of vertical sustainability via IP rights is therefore to provide enough control to prevent wholesale appropriation on the one hand, while at the same time providing enough access to an author's work to ensure a steady supply of non-proprietary source code to sustain open source efforts. When this balance is properly struck, open source contributors can be assured that their work will be neither stolen from below nor strangled from above.[68]

IV. WHAT KIND OF OPEN SOURCE IP IS RIGHT FOR YOU?

In the preceding Parts, I have discussed the ways in which an IP rights system properly focused on attribution and vertical sustainability can incentivize open source production while also maximizing the quality of work produced. Having established the theoretical framework of attribution and vertical sustainability as the basic tenets of a successful open source IP system, this Part will critically examine several of the IP implementations currently in use and evaluate their success and functionality through this lens.

In general, all open source licenses are designed to allow software developers to relinquish some traditional legal copyrights while retaining others. Some licenses, like the MIT license,[69] do this by simply disclaiming a large portion of the rights provided by American copyright law. Other licenses, like the GNU General Public License (GPL),[70] take a more

---

2005). I feel this attitude is not only shortsighted, but fundamentally counterfactual. I address copyleft licenses in Part IV of this article.

68. Professor Wagner directly rebuts those who argue no such balance is possible and that any form of IP control will eventually strangle open source initiatives. *See* Wagner, *supra* note 20, at 1030-31.

69. Open Source Initiative, The MIT License, http://www.opensource.org/licenses/mit-license.php (last visited Nov. 5, 2005).

70. Open Source Initiative, The GNU General Public License (GPL), http://opensource.org/licenses/gpl-license.php (last visited Nov. 5, 2005).

radical position. Through a limited surrender of rights, they attempt to provide the public with greater freedom of access to open source works while at the same time sustaining the open source community by applying strict terms of use to the licensed software.

Because all open source licenses attempt to allow the author(s) and the public to interact in a way that normal copyright does not, they are an inherently imperfect solution to what many individuals see as the unreasonableness of contemporary copyright law. Significantly, the narrow question of the enforceability of these licenses has never been directly challenged in the American courts.[71] It is nevertheless instructive to question what the goals of these licenses are and how they attempt to reach them.

A. Attribution

The one trait all licenses have in common is some requirement of attribution. The de minimis language is usually something like "the above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software."[72] This requirement ensures that even as software is passed on and modified from user to user, each receiving individual is aware who they are receiving the software from and what terms that party has imposed.

Version 1.1 of the Apache Software License,[73] in particular, is closely focused on proper and permanent attribution. Not only does the license require that all redistributions and derivative works of the software bear the sentence, "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)," but it also stipulates that "Products derived from this software may not be called 'Apache', nor may 'Apache' appear in their name, without prior written permission of the Apache Software Foundation."[74]

These two clauses represent the heart of attribution with regard to open source software. In the first clause mentioned above, Apache ensures that no matter how many generations of development, modification, and redistribution their software and its derivative works go through, each user will be notified that at least some of the code in the program they receive was

71. *See infra* note 86 (discussing software license terms as litigated in Germany).

72. Open Source Initiative, The MIT License, http://www.opensource.org/licenses/mit-license.php (last visited Nov. 5, 2005).

73. The Apache Software Foundation, Licenses, http://www.apache.org/licenses/ (follow hyperlink under "Apache License, Version 1.1 (historic)") (last visited Dec. 6, 2005). This version of the Apache license was replaced in January 2004 with version 2.0. *Id.* (follow hyperlink under "Apache License, Version 2.0 (current)").

74. *Id.*

originally developed by Apache. This permanent attribution is critical if open source is going to be incentivized through maximizing *H* and *SP* type rewards.

The second clause mentioned above plays an even more important attribution role. That clause allows derivative works to be produced only on the condition that they are clearly marked as something other than the original Apache software. Apache thereby ensures two critical attribution traits. First, the public will not be fooled by using software they think is from Apache but is in fact an unreliable third-party modification of Apache software. Such accurate attribution preserves the public goodwill Apache can cultivate by producing quality software and incentivizes that production by placing Apache in competition with works derivative of its own source code. Second, only downstream modifications approved by Apache may carry the Apache name. This encourages downstream developers to implement and modify the original Apache code in a way that improves upon it sufficiently such that Apache is willing to license its name to that product.

Other licenses take attribution equally seriously. Condensing the Apache language into one clause, the Academic Free License v. 2.1 reads in part:

> Attribution Rights. You must retain, in the Source Code of any Derivative Works that You create, all copyright, patent or trademark notices from the Source Code of the Original Work, as well as any notices of licensing and any descriptive text identified therein as an "Attribution Notice." You must cause the Source Code for any Derivative Works that You create to carry a prominent Attribution Notice reasonably calculated to inform recipients that You have modified the Original Work.[75]

Again, by mandating accurate attribution and notice of modification, the Academic Free License seeks to ensure that the developers who use the license receive credit for their work's successes and are held responsible for its failures, while at the same time allowing the public to make informed and accurate decisions about whether to use a product based on its pedigree. The ability of any license or IP rights regime to incentivize the production of quality goods is fundamentally tied to its attribution function. That nearly every legitimate open source license in use today has at

75. Open Source Initiative, The Academic Free License v. 2.1, http://opensource.org/licenses/afl-2.1.php (last visited Nov. 5, 2005).

least a rudimentary attribution clause is evidence of the crucial role of attribution in open source development.[76]

B. Copyleft and Vertical Sustainability

While there are many so-called open source licenses available to software developers, I divide these licenses into two general groups: copyleft[77] and non-copyleft licenses.[78] At the minimum, all open source licenses allow the licensee to access the source code of the program and copy and manipulate its contents. Where copyleft and non-copyleft licenses diverge is on the terms of redistribution. The license terms of copylefted programs require that any derivative works of the copylefted program be released under identical license terms, whereas derivatives of non-copylefted programs may be redistributed under any license terms.[79] The philosophy is one of forced vertical sustainability: by explicitly requiring any software utilizing copylefted open source code to itself be copylefted, the pool of copylefted open source software will necessarily expand. This is the motivation behind the argument raised by many open source opponents that open source software is "viral" in nature, because once copylefted code is included in a program, even a small bit, the li-

---

76. Of the fifty-eight different open source licenses listed as "approved" by the Open Source Initiative at http://www.opensource.org/licenses, nearly every single license has at least a basic attribution requirement. These requirements range from the explicit:

> 2. Redistributions of the Code in binary form must be accompanied by this GPG-signed text in any documentation and, each time the resulting executable program or a program dependent thereon is launched, a prominent display (e.g., splash screen or banner text) of the Author's attribution information, which includes:
>
> (a) Name ("AUTHOR"),
>
> (b) Professional identification ("PROFESSIONAL IDENTIFICATION"), and
>
> (c) URL ("URL").

Open Source Initiative, Attribution Assurance License, http://www.opensource.org/licenses/attribution.php (last visited Dec. 6, 2005), to the more general requirement that all copyright information of previous versions must be preserved in subsequent versions, to the very lax requirement that all versions must be tracked via different version numbers and acknowledgement of previous versions. *See* Open Source Initiative, Licenses, http://www.opensource.org/licenses (last visited Dec. 6, 2005).

77. Copyleft licenses include the GNU GPL, the French CeCILL license, the Affero GPL, and the OpenSSL license.

78. Non-copyleft licenses include the MIT license, the BSD license, the X11 license, the Apache license, and almost all proprietary software licenses.

79. To be more precise, the non-copyleft license itself usually determines what license terms apply to derivative works. Only licenses that require derivative works to be distributed under license terms identical to the original license are considered copyleft.

cense terms of that copylefted code dictate that the entire program must now be distributed under identical copyleft license terms.[80]

The GNU[81] GPL is the most famous copyleft license and has become the de facto standard.[82] Section two of the GPL reads in part:

> 2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
>
> . . . .
>
> b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.[83]

This type of mandatory license terms, which require those who wish to use open source software to only distribute their work under open source terms, has legitimate drawbacks. To understand these drawbacks, a little background information is necessary.

Expanding on the excerpted section above, section two, paragraph two of the GPL goes on to say:

> These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permis-

---

80. The term "viral," as applied to software, is extremely content laden. Although a technically accurate description of a license that is recursively self-propagating, the term has been propagandized by copyleft opponents trading on the unrelated connotation of viral as unauthorized code that damages computer systems. *See* Craig Mundie, Remarks at The New York University Stern School of Business: The Commercial Software Model, (May 3, 2001), *available at* http://www.microsoft.com/presspass/exec/craig/05-03sharedsource.asp.

81. GNU is a recursive acronym for GNU's not UNIX.

82. For an in-depth discussion of the GNU GPL's place in the open source debate, see David McGowan, *Legal Implications of Open Source Software*, 2001 U. ILL. L. REV. 241 (2001).

83. Open Source Initiative, The GNU General Public License (GPL), http://opensource.org/licenses/gpl-license.php (last visited Nov. 5, 2005).

> sions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.[84]

This section of the GPL is perhaps the most controversial. There has been much debate on the issue and many argue that this clause holds that open source developers who wish to commercialize their products, as well as downstream developers who have modified an open source program, are unable to do so under the GPL.[85] While this contention has never been litigated to conclusion,[86] it is likely that in reality the above section does serve as a significant, although not complete, practical bar to the use of GPL-ed code in conventionally commercial applications.[87]

This bears further significance in the arena of software development, where reusing code proven to be reliable and efficient is an important means to secure efficiency.[88] Imagine that there is a particularly graceful

---

84. *Id.*

85. Richard Epstein has discussed this and other arguments against the GPL with James Boyle in a point-counterpoint format. *See* Richard Epstein, *Why Open Source is Unsustainable*, FIN. TIMES ONLINE, Oct. 21, 2004, http://news.ft.com/cms/s/78d9812a-2386-11d9-aee5-00000e2511c8.html#U101244209021g4.

86. In April 2004, a German court ruled on the narrow question of whether the GPL was a legally enforceable license, holding that the defendant had:

> infringed on the copyright of plaintiff by offering the software "netfilter/iptables" for download and by advertising its distribution, without adhering to the license conditions of the GPL. Said actions would only be permissible if defendant had a license grant. . . . This is independent of the questions whether the licensing conditions of the GPL have been effectively agreed upon between plaintiff and defendant or not. If the GPL were not agreed upon by the parties, defendant would notwithstanding lack the necessary rights to copy, distribute, and make the software "netfilter/iptables" publicly available.

Perhaps most telling is the fact that no company that attempted to circumvent the GPL has not backed down when threatened with suit by the copyright holder or the Free Software Foundation (the body that holds the copyright in the language of the GPL and administrates its use). Thus while lacking so far in *de jure* enforcement the GPL is a strong *de facto* deterrent to infringing the terms it contains. *See* GNU General Public License, WIKIPEDIA: THE FREE ENCYCLOPEDIA, http://en.wikipedia.org/wiki/GNU_General_Public_License (last visited Dec. 6, 2005).

87. While the terms of the GNU GPL prohibit selling software itself, there is no prohibition on distributing GPL'd software for a fee. In fact, Richard Stallman encourages it. *See supra* note 55. Nevertheless, copyleft licenses like the GPL prevent developers who incorporate copylefted code into their projects from determining their own license terms for those projects. There are many reasons a developer might not want to distribute her work under the GPL. Unfortunately, by the licenses' own terms, downstream rejection of the license de facto requires immediate rejection of the code.

88. *See supra* note 20 (discussing the widespread practice of reusing old code in new projects).

and well-known bit of code that performs its function exceptionally well and is licensed as open source under the GPL. Developers who wish to use this code as an integral part of a larger program would be bound to license that larger program under the GPL as well. If they didn't want to be held to the terms of the GPL, then they would be barred from using the code.

Thus, the fact that GPL-ed code is distributed as an integrated software package that forces the whole to be licensed under the GPL is counterproductive in two important ways. First, it deters software developers from implementing GPL-ed code in non-GPL projects. In other words, by trying to render code more accessible by distributing it under an open source license, the GPL in fact brings about the same stifling results as strict IP control for developers who want to integrate GPL-ed code but do not want to be held to distributing it under the GPL. This is because that code is essentially as unavailable as it would be if it were conventionally copyrighted.[89]

Second, if open source is to legitimately compete with proprietary production it needs to be flexible enough to be a means to an end rather than an end in and of itself. That is to say, there is no reason that commercial projects cannot be developed in an open source fashion.[90] The strength of open source is in the ability of every user to adapt the software to her specific needs, and open source licenses should reflect this crucial characteristic. An open source license that does not allow developers to alter its terms upon modifying and redistributing the source code runs the significant risk of falling prey to the same transaction costs and institutional barriers endemic to proprietary software development and that instigated the open source movement in the first place.[91]

---

89. *See* McGowan, *supra* note 82, at 255-60.

90. Indeed, it is the aim of this Article to examine some of the ways in which IP rights might aid such an effort.

91. Placing this problem with copyleft licenses like the GPL in more concrete terms, Tom Walker has analogized the effects of the GPL and strong traditional copyright protection:

> When I buy music protected by DRM, the seller intends to stop me from making copies of songs. When I use software that is licensed under the GPL, the developer intends to stop me from making the software "closed," or non-free. The intentions obviously aren't even slightly similar, but the consequences are. Both the GPL and DRM want to restrict my use, and re-use, of the music or source code that I obtain. Neither does what it was designed to do. Both have unintentional harmful effects.

Tom Walker, Toward True Open Source (July 16, 2004), http://software.newsforge.com/-software/04/07/15/163208.shtml.

C. Value-Added Quality

Nevertheless, the notion of copyleft is not entirely inconsistent with the goal of incentivizing quality open source production. In fact, one area where it provides significant gains is in improving the quality of software produced. Because of the barrier to outright commercialization that copyleft licenses like the GPL represent, developers that choose such a route are highly incentivized to provide rich value added features that may be sold for a fee even under copyleft licenses.

Red Hat, the most popular version of the highly competitive Linux operating system, is often held up as an admirable example of how a complex software suite developed through open source can secure impressive commercial success. Further, even though Red Hat Linux is largely distributed under the GPL, it costs between $25 and several hundred dollars per "subscription."[92] While the Linux operating system is licensed under the GPL and therefore cannot be commercially sold, Red Hat successfully competes with Microsoft and other large software producers by undertaking and charging for significant value addition.[93]

This value addition takes place in two primary ways. Red Hat is not actually charging for the Linux operating system, but instead for comprehensive technical service as well as non-GPL-ed software bundled with the operating system in the Red Hat distribution of Linux. For instance, an individual user can purchase the Red Hat package for $179 and receive a guarantee that her hardware will be compatible with the software, live, around the clock tech support, and a slew of on-site training options.[94] In contrast, a no-frills version of the Linux operating system is available for free, and without any of the value addition options sold by Red Hat, from the Fedora Project.[95] Open source developers who produce under copyleft licenses must commit themselves to value addition if they wish to receive payment for their software. By prohibiting direct commercialization, copyleft licenses do a very good job of incentivizing quality-maximizing

---

92. The price depends on which Red Hat distribution you purchase. Student desktop software suites start at around $25, whereas full corporate implementations can cost several hundred dollars. *See generally* Red Hat, http://www.redhat.com (last visited Nov. 29, 2005).

93. When I say "competes with Microsoft" I of course do not mean that Red Hat has captured more than a small fraction of Microsoft customers. However, it is interesting that an open source effort such as Red Hat is competing at all.

94. *See* Red Hat, Red Hat Enterprise Linux: The Corporate Linux Standard, http://www.redhat.com/en_us/USA/rhel/ (last visited Dec. 6, 2005).

95. *See* Fedora, Red Hat Enterprise Linux, http://fedora.redhat.com/about/rhel.html (last visited Dec. 6, 2005) (comparing the Fedora Project to Red Hat Enterprise Linux).

value addition even as they tend to deincentivize the use of copylefted code by open source developers not producing in a copyleft environment.

V. A THEORETICAL ALTERNATIVE

Having identified and discussed the basic elements of a successful open source IP system and the general attributes of a few open source licenses, this Part will propose a theoretical alternative to current open source licenses and conventional IP regimes. As discussed in this Article, a properly applied IP rights regime can incentivize and maximize the quality of open source production by minimizing transaction costs and focusing on proper attribution and vertical sustainability. To meet these goals, an open source IP system should minimize procedure (and thereby minimize transaction costs) while maximizing *SP*, *H*, and, if possible, *M* type rewards. Any ostensibly open source procedure not narrowly focused on attribution and/or vertical sustainability runs the serious risk of remaining a vestige of twentieth-century IP regimes focused on rights of exclusion rather than enabling and encouraging rapid software development.

A. What We Don't Want

If an IP system is to maximize incentives for production, it makes little sense to build in restrictions against commercializing the software that prevent direct economic competition with proprietary developers. Therefore, there should be no restrictions placed on the commercialization of source code modified and/or combined with other code in a non-trivial way. In other words, copyleft type requirements, whereby downstream software that implements open source code must itself be released as open source software, should be avoided.

Further, in order for developers to harness the full potential of the open source model they must be able to fashion license terms that they feel will best suit their particular business and development model. For that reason, developers should be free to apply whatever license terms they wish to the non-trivial derivative modifications they make to open source code. Allowing developers to either tighten or relinquish the control they have over their own work will incentivize their participation in the open source community and allow developers to check or accelerate the distribution and modification of their own code without affecting the work on others' code.[96]

---

96. Linus Torvalds, the originator of Linux, has himself said, "My opinion on licenses is that 'he who writes the code gets to choose his license, and nobody else gets to complain.' Anybody complaining about a copyright license is a whiner." E-mail from

In order to minimize the transaction costs related to securing rights, copyright—rather than patent or trade secrets—should be the mode of protection for computer programs. Because of the theoretically infinite duration of protection, trade secrets are antithetical to open source production. Although some court cases from the early days of the computer age wrestled with the idea/expression dichotomy as it applies to computer programs,[97] it is now settled law that computer code is copyrightable, whereas implementations and utilizations of computer code are available for patent.[98] Further, whereas obtaining a patent is a long, complicated, and often very expensive procedure,[99] copyrights are granted automatically at the moment of creation and are easily licensed. Copyrights (as opposed to patents or trade secrets) are therefore the best mode of protection for software in general, as they strike a decent balance between flexibility, accessibility, and protection.

Unfortunately, the duration of copyright protection is entirely too long to be applicable to the software development industry. The biggest necessary departure from current IP law is therefore the drastic shortening of the duration of protection for all software, both open and closed source. The

---

Linus Torvalds to Steve Hutton (July 10, 1998 11:00:21 PDT), *available at* http://linuxtoday.com/news_story.php3?ltsn=2000-09-05-001-21-OP-LF-KE&reply=00172&quote=1.

97. *See* David A. Einhorn, *Copyright and Patent Protection for Computer Software: Are They Mutually Exclusive?*, IDEA 265, 266 (1990). Einhorn writes:

> The Copyright Office began registering copyrights on computer software in 1964. Registration by the Copyright Office, however, raised only a presumption in favor of copyright validity. This presumption was vulnerable to challenge in the courts. Any doubts as to whether software could be the subject of copyright were dispelled by the passage of the Computer Copyright Act of 1980. According to the House Report, the effect of this brief amendment was to 'clearly [apply federal copyright law] to computer programs.'

Id.

98. Diamond v. Diehr, 450 U.S. 175, 188 (1981) (holding that otherwise patentable processes implemented via computer program are themselves patentable).

99. Depending on the nature of the invention and the extent of the prior art search required, obtaining a United States patent can cost anywhere from $5,000 to $10,000. Further, the prior art search typically can take anywhere from three to six weeks, drafting claims and preparing the application usually takes six to eight weeks, prosecution can take a year or two, and the issue process is typically three to nine months. If everything goes according to plan, it commonly takes twenty-four to thirty-six months to secure an average patent. *See generally* U.S. Patent and Trademark Office, Frequently Asked Questions, http://www.uspto.gov/main/faq (last visited Dec. 6, 2005); Markets, Patents & Alliances L.L.C., Answers to Commonly Asked Questions About Patents, http://www.marketsandpatents.com/faq.html (last visited Oct. 17, 2005).

current term of copyright protection, the life of the author plus seventy years,[100] might as well be infinite in the world of software development. Rarely is software relevant and almost never is it commercially viable after more than a few years. Shortening the term of protection to a decade or less would, therefore, both stimulate the pace of production while ensuring a healthy body of open source resources.

B. What's Left?

Now that I've identified a few major traits that an open source IP system should not have, what are we left with? Clearly the most fundamental aspect of any open source IP regime is required access to open source code. Access to the code should be as easy as possible and the code should be made available in forms that facilitate maximum comprehension and utility. This most basic requirement lays the groundwork for legitimate vertical stability and, as such, should be most protected by the IP system. Second, if open source is to be anything more than a distribution scheme, developers must have the right to copy and manipulate the source code as well. The source code must be made available for combination with any other type of code so that developers can adapt the code to their own uses.

The necessary corollary to the right to copy and manipulate source code is the right to distribute both the source code and any modified version of the code as a derivative work. However, as mentioned above, this right to distribute should not be an obligation. The terms of distribution for any derivative work should be set by the author of the derivative work, not of the original source code.

When combined, these three aspects of an IP system will strike the critical vertically sustainable balance between providing developers with enough control over their work to incentivize production, and at the same time ensure that a sufficient pool of open source code remains available to sustain development.[101] Securing rights of access, manipulation, and distribution for developers while allowing them the freedom to determine under what terms downstream distribution will take place provides the greatest opportunity to maximize *H* and *SP* type rewards. At the same time, this approach harnesses the incentives towards open source production these rewards create to spur production.

Vertical sustainability will be severely handicapped, however, if not paired with a viable attribution mechanism. The guarantee of access, ma-

---

100. 17 U.S.C. § 302(a) (2000).

101. *See* Wagner, *supra* note 20 (discussing the balance a proper IP regime strikes in terms of benefits to creators and sustainability of the system).

nipulation, and distribution rights must be balanced by the firm obligation of proper attribution at every stage. All source code must be accurately and clearly attributed to the responsible individual(s). Additionally, that attribution must be preserved in every distributed copy of the unaltered code as a prerequisite for the right of distribution. Finally, any code that is modified in any way must be clearly identified as such, with the original author attributed, the modifying author identified, and a clear explanation of the nature and extent of the modification. Thus, each dynasty of open source software will have a clear provenance and be of maximum utility to the community.

And that's it. Building in more restrictions or complicating the system will only raise transaction costs and disincentivize production. For an open source IP system to work, it must be simple, adaptable, and fluid—just as the open source development process itself.

C. The Middle Path: A Way Out

It is crucial to recognize that all software licenses, even the very best, are vitally and intrinsically flawed as instruments of open source development. Because they are, in fact, contracts between the developer and the consumer, the software licenses that fuel the open source movement do so by invoking contract law as a stopgap measure in an attempt to patch what open source developers see as a broken copyright jurisprudence. However, by piling a second layer of law on top of a notoriously complex intellectual property regime, these licenses drive transaction costs through the roof. The spiraling transaction costs are exacerbated by the fact that the enforceability of these licenses is expensive at best and impossible at worst.[102] Further, since each license and the rights it purports to grant are different, it is difficult for any individual or group to ascertain what allowance and obligations they have accepted, especially when many differently licensed open source software are combined in a single project.

Current copyright law grants the copyright holder[103] the *exclusive* right to make and sell copies of the work, to import and export the work, to make derivative works, to publicly perform the work, and to sell or assign these rights to others.[104] Many open source advocates argue that this method of assigning rights fails because of the crucial exclusivity of the

---

102. For a discussion of some barriers to successful open source litigation under a license regime, see LAWRENCE ROSEN, OPEN SOURCE LICENSING 269-94 (2004).

103. This is usually the author, since copyright vests immediately upon creation of the work. However, due to licenses or other contractual terms, copyrights are often held by someone other than the author.

104. *See* 17 U.S.C. § 106 (2000).

assignment. If the author is the only one allowed to make and sell the work and its derivatives as well as transfer the rights then there is no way for the type of rapid and widespread distribution, access, and contribution that open source development relies on to get off the ground. The transaction costs are just far too high.

Attempting to solve this fundamental transaction cost issue by resorting to an additional layer of contract law in addition to copyright is simply swapping one type of transaction cost for another. Instead of a better license, a better approach is required.

Indeed, the last time American copyright had a major overhaul was 1976.[105] This revision was largely motivated by and a reaction to the budding technology age and the first few years of widespread computer and software utilization. This precedent of adjusting copyright laws so that their practice conforms to their purpose should be followed today—the law must again be changed to encourage rather than punish open source producers.

The change need not be drastic or universal. Conventional copyright works very well for the vast majority of copyrightable works. All that is necessary is to make a simple choice available to software developers. As the law now stands, software developers (regardless of their production model) have an empty choice: accept conventional copyright protection (and then alter their rights through license terms if necessary) or secure no protection at all. Rather than this "damned if you do and damned if you don't" approach, software developers should be able to choose between copyrighting their works, and "opening"[106] their works. Openrighted works would be copyrighted in a sense, but would be subject to very different types of rules. Clearly, the simpler the rules the better, as the purpose of opening your work rather than copyrighting would be to retain enough control to incentivize production of quality software as well as protecting the open source model itself.

---

105. This fourth major revision to the U.S. Copyright Act introduced several changes, and retroactively preempted all other U.S. copyright law. Many of these changes were in response to new technologies such as computers and audio recording formats. Additionally, the 1976 revision adjusted U.S. copyright such that America's law conformed more with international norms in preparation for the U.S. becoming a Berne Convention signatory. *See generally* CRAIG JOYCE ET AL., COPYRIGHT LAW 22-28 (5th ed. 2001).

106. I use this term to facilitate my argument. The name of the sub-category of copyright I advocate for creating here is probably not important. Any number of names would work; this one is chosen purely for simplicity.

A statutory framework that incorporates the above ideas might look something like the following:

1) Anyone has the right to:
   a) access the source code of this software
   b) manipulate the source code in any way, and
   c) distribute the source code of this software and any derivative work based on the source code of this software for any purpose.

2) If this software is distributed unchanged, the unaltered source code, and only the unaltered source code, will remain under the terms of this statute.
   a) If this software is combined with any other software or code, the terms of distribution for the portion of the combined software made up of that other software or code are not determined by this statute. The portion of the combined work that is identical to this software is still controlled by this statute.
   b) If this software is altered in any non-trivial way, the terms of distribution for the derivative work embodied by the altered software are not determined by this statute.

3) The rights granted in sections 1 and 2 are contingent upon preserving the clear attribution notice in the source code of the software.
   a) If this software is combined with any other software in any way, notice must be provided of the original author(s) of this software, notice of the combining party's identity, and notice that this software has been combined with other software. If reasonably possible, notice of the author(s) of the other software must be provided as well.
   b) If this software is altered in any way, notice of the original author(s) of this software, notice of the altering party's identity, and notice that this software has been altered must be provided.

Such a set of "open" rules, focused closely on attribution and vertical sustainability, allows for enough control to incentivize open source development while at the same time maintaining the decentralization and low transaction costs that open source development requires. Section 1 covers the basics, allowing anyone to access and manipulate the source code of

openrighted software, and to create and distribute any derivative works they like. Section 2 deals with vertical sustainability. The terms are explicitly non-copyleft. Section 2(a) makes clear that open source code that comes into a project and is combined with other code but is itself unaltered remains open source without affecting the code with which it has been combined. In other words, open source code that is used in a project in an unaltered form will remain open source. The author is free to determine her own license terms under traditional copyright or to maintain the work's open status for all other code in the project, whether proprietary code or code that was once openrighted but has been non-trivially altered.

This provision plays three important roles. First, it prevents unilateral downstream appropriation because open source code can only leave the public domain under these rules if it is altered in a non-trivial way. Even if open source code used under these rules is combined with other code outside the scope of this license, the portion of the resulting program that was originally open source will remain so, even as the code it has been combined with will remain untouched. That means that open source code can be used in proprietary commercial projects without reducing the pool of open source software. Second, such control over the fruits of one's labors is an important way to incentivize open source. It makes sense from a Lockean perspective that the author should be allowed to determine how to dispose of code she has modified herself even if the code was originally open source.[107] Finally, Section 2 acts in concert with Section 1(c) to allow the ready integration of open source solutions to commercial applications. Because everything a developer builds herself or procures from other sources is explicitly exempt from these rules (no matter how it is used in conjunction with software under these rules), there is no reason why implementing openrighted software into a larger project would hinder the developer from protecting the larger project in any way she sees fit.

Clearly such ease of implementation that maintains vertical sustainability can have significant incentivizing effects on developers. These rules can further promote open source development through the compre-

---

107. Lockean theory, also known as the "sweat of the brow" justification, has long been intertwined with American copyright. This theory of property, which holds that people should benefit from the fruit of their efforts, is fundamental to the quid pro quo the constitution establishes for inventions and creative works. Under such a Lockean theory, if a developer works to create software that happens to include some open source code, that innovating developer should have the right to deploy her software under whatever terms she sees fit. For a more complete explanation (and criticism) of Lockean copyright theory, see Carys J. Craig, *Locke, Labour and Limiting the Author's Right: A Warning Against a Lockean Approach to Copyright Law*, 28 QUEEN'S L.J. 1 (2002).

hensive attribution requirements of Section 3. These requirements, by ensuring that the pedigree and identities of all who worked on software are obvious at all stages of development, would encourage the production of quality software. Additionally, because any developer who implements code under these rules must give explicit notice that she has done so, whether through combination or alteration, such stringent attribution requirements also serve an incentivizing function whenever open source code under these rules is used in a larger project. Put another way, if some subroutine of a larger program works exceptionally well, anyone who cares to look can easily determine who was responsible for that portion of code.[108]

Finally, the "non-trivial" language in Section 2(b) serves two critical purposes. First, by exempting only non-trivial alterations of the software from the openright rules, the terms of the rules strike an important balance between prohibiting unilateral appropriation through superficial modification of the software and ensuring that developers who legitimately modify the software are free to redistribute the fruits of their labors on any terms they wish. Second, and more importantly, "non-trivial" is a largely subjective term. In some instances a non-trivial alteration might be deleting a vestigial line of code, whereas in others the alteration of a single character might be critical. This built-in subjectivity allows for case-by-case judicial interpretation of the application of the openright rule terms. Rather than rigid license terms that can only break if stressed enough, these rules are deliberately adaptable so that they can give, bend, and apply to a far greater number of instances.

It is important to reiterate that, as I conceive it, this openright regime would not replace copyright, but supplement it as an alternative to traditional copyright only available for software. Further, it is crucial that the possibility that an author could inadvertently openright her software when she meant to copyright it be extremely small. Fortunately, current copyright law facilitates a solution. Conventional copyright vests in the author at the moment of creation. As soon as software is created, it is automatically copyrighted to the author (or her contractual proxy).[109] Because openrighting her software means that a developer would secure fewer rights than through copyright, copyright should remain the default condition. That means that an extra positive step should be required to openright software. This could very easily be facilitated with negligible transaction

108. *See* Maher, *supra* note 13, at 631-36 (discussing the importance of attribution); Benkler, *supra* note 11, at 423-26 (same).

109. *See* 17 U.S.C. § 106 (2000).

costs if the United States Copyright Office were to maintain a database of all openrighted code. If a developer wanted to traditionally copyright (and perhaps later license) her software, she would be free to follow the same procedures as are used today. However, if an author wanted to openright her software, she would be required to file a copy of the source code of that software with the Copyright Office to be archived in the database.

Such an archival step would serve three crucial purposes. First, it would serve the previously mentioned preventative function, ensuring that no software became openrighted without the author's consent. Second, the archive would serve a crucial vertical sustainability function by maintaining a permanent repository of open source code—an effective one-stop shop for the open source developer. Because the act of electronically transmitting code to a government server is identical to that of transmitting to any other server, the required filing with the Copyright Office would not increase transaction costs. Finally, the archive would greatly simplify and facilitate enforcement of the openright rules because the repository would serve as a collection of all "official" openrighted code. Because alteration and combination of openrighted software are central to the openright rules and their functions, practical enforcement would require an "official" copy of any code against which disputed code could be compared. Keeping a copy of all openrighted code in one place would critically minimize the transaction costs involved in this procedural necessity.

Such a streamlined, centralized system explicitly designed around accountability and vertical sustainability would drastically reduce transaction costs and, thus, not only directly incentivize the production of quality open source software but represent a very real improvement over the current copyright system. The number of copyright/license agreements that developers use today creates a "briar patch" of transaction costs. Centralizing the system under the most minimal and voluntary terms available would sharply reduce the transaction costs that inhibit open source development because actors will comprehend the rules, enforcement will be simplified, and transparency will be facilitated. Importantly, because the openright system I suggest would be a completely voluntary alternative to existing copyright statutes, what was once a broken system restraining the open source model's development into a fully competitive production practice would be rejuvenated into an effective incentivizing and rewarding mechanism with little effect on those who were happy with the system the way it was before.

VI. CONCLUSION: IT'S TIME TO CALL A TRUCE

The battle between the equally shrill champions of open source and protectors of intellectual property must end. In the end, we all want the same thing: more software that works better. As a production mode, open source development is neither the savior nor the damnation of software development. Rather, it is simply a different way of achieving the same goal as the traditional intellectual property enthusiasts. In many instances, the personality many advocates of both sides have invested in their arguments have overshadowed this point.

Regardless of what side of the debate you are on, the growing list of open source successes should make it clear that, at the very least, open source can be a powerful mode in which to develop software. In a rational society that wants more software that works better, it makes little sense not to encourage any capable production mode. That certainly does not mean that open source should be the only mode, it merely means that it does not make sense to squash it completely with strict traditional property controls and prohibitively high transaction costs. In this battle, as in most, the middle path is the one that makes the most sense and will result in the most good for everyone.

Intellectual property, properly and carefully deployed, can be an important way to incentivize and optimize open source production. Copyright protection has been successful in being a significant motivator for many to create. Yet, because creation is fundamentally an act of compilation, the more resources available to the author, the more likely she will be able to create something wonderful. Solutions like the one presented here lie at the nexus of these two principles and it is this type of middleground solution that will be necessary to enable the next century of software development.