Open Source Software: The Success of an Alternative Intellectual Property Incentive Paradigm

Marcus Maher*

INTRODUCTION

Intellectual property protection in the United States is based on an incentive system.[1] The protections provided by intellectual property law are designed to produce economic incentives to "promote the progress of science and useful arts."[2] The open source software movement, which has gained publicity as the popularity of the Linux operating system has grown, provides an alternative to the economic incentives that dominate the thinking of U.S. intellectual property policy. Product comparisons show that open source software attains high technical standards despite the relative absence of economic motivation for the creators of this software. This technical success is all the more puzzling to the traditional computer community, given the distributed, almost ad hoc, development methodologies employed by the open source movement. This article shows how the science of complexity theory is able to explain the open source movement's ability to trans-

* Associate, Wiley, Rein & Fielding, J.D., Harvard Law School. The views expressed in the article are entirely those of the author. The author thanks Prof. Lawrence Lessig for helpful comments and criticism.

1. *See Statement of Copyright and Intellectual Property Law Professors in Opposition to H.R. 604, H.R. 2589, and S.505 "The Copyright Term Extension Act"*(Submitted to the Committees on the Judiciary United States Senate United States House of Representatives)(last modified Jul. 16, 1999) <http://www.public.asu.edu/~dkarjala/legmats/1998Statement.html>.

2. U.S. CONST. art. I, § 8.

late non-economic incentive mechanisms into a process for technological development and innovation.

This paper will begin to address this quandary by providing a factual introduction into the details of the open source development process. Next, a background introduction to complexity theory will be provided. The features of open source development will then be analyzed, uncovering the complex nature of open source development. While the complex nature of open source software provides an explanation as to its technical success, it also provides insight into a number of problems that are facing the open source community. The threats to the complex nature of open source development will be considered and means of circumventing these problems suggested. Finally, the potential for complexity to solve some anticipated open source problems will be discussed.

I. OVERVIEW OF OPEN SOURCE SOFTWARE DEVELOPMENT

The open source software development process has been described as similar in nature to the familiar methods of research in the scientific community.[3] Key to the scientific method are the principles of discovery and justification. Justification of scientific discoveries comes from peer review. This review is only possible if the discovery process is shared - the hypothesis, the experiment, the analysis, et cetera. The sharing of ideas allows not only vindication of the initial results, but information upon which other scientists can build, allowing advancement of the state of knowledge and the strengthening of existing theories.[4]

Open source methods could be seen, in part, as the scientific method at work in the computer science community. As will be seen, however, the reality is more complicated. The principles of both discovery and justification play extremely important roles in delineating the bounds of activity in the open source community.

3. *See* Chris DiBona et al., *Introduction*, *in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION 1, 6-7 (Chris Dibona et al. eds., 1999).

4. *See id.*

A. *Open Source Examples*

Before describing the development process, it is useful to have a basic understanding of a few of the more prominent programs and packages that have been developed through the open source process.

1. Apache

Apache began as an effort to address problems that were perceived in the NCSA httpd web server.[5] The Apache has been the most popular web server since April, 1996 and today is more widely used than all other web servers combined.[6] The advantages of Apache include the fact that it is free (as in costless), that it is free (as in open source), and that it provides high quality performance.[7]

2. BIND

The Berkeley Internet Name Domain package (BIND) is the software that provides domain name service (DNS)[8] for the vast majority of name serving machines on the Internet. BIND was originally developed at Berkeley under a grant from the Defense Advanced Research Projects Agency (DARPA)[9]. However, the development and maintenance of BIND has been taken over by the Internet Software Consortium (ISC).[10]

5. *See About the Apache HTTP Server Project* (visited Apr. 20, 2000) <http://www.apache.org/ABOUT_APACHE.html>.

6. *See id.*

7. *See id.*

8. *See ISC BIND* (visited Jan. 27 2000) <http://www.isc.org/products/BIND>

9. *Id.*

10. *See ISC BIND* (visited Jan. 27 2000) <http://www.isc.org/products/BIND>.

3. Linux (GNU/Linux)[11]

Linux is a Unix-like operating system. It is not, however, a version of Unix, based on a version of Unix from the basic AT&T Unix sourcecode, or a derivative thereof.[12] It was intentionally modeled after Unix to make it convenient for others to adopt,[13] and because it had been proven to be portable.[14] Beginning in 1984, the GNU Project worked to create the elements of a complete operating system.[15] By the early 1990s the only substantial operating system element remaining to be developed, was the kernel.[16]

The Linux kernel started as the hobby of Linus Torvalds, a Finnish computer science student, in part to teach him about his new 386 computer.[17] He started with the "Minix" kernel, a kernel created by a computer science professor for educational use. Torvalds eventually rewrote the entire kernel, creating the foundation for the Linux kernel. After the kernel was mentioned on a Minix newsgroup, he was provided the opportunity to distribute the kernel publicly on an FTP server.[18]

The Linux kernel was the final element needed for the operating system. With some effort, the kernel and existing GNU programs were integrated into a functioning operating system.[19] The

11. Although this open source operating system is most commonly referred to as "Linux," there have been some recent efforts, in particular by Richard Stallman, to encourage use of the name "GNU/Linux," due to the substantial role of GNU software in the operating system. *See* Richard Stallman, *Linux and the GNU Project*, (last modified Jan. 22, 2000) <http://www.fsf.org/gnu/linux-and-gnu.html>. Since the term "Linux" is still the most common usage, that term will be used throughout.

12. *See* Linus Torvalds, *The Linux Edge*, *in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION, *supra* note 3 at 101,102.

13. *See* Richard Stallman, *The GNU Manifesto* (last modified Jan. 16, 2000) <http://www.fsf.org/gnu/manifesto.html>.

14. *See* Richard Stallman, *Overview of the GNU Project* (last modified Nov. 2, 1999) <http://www.fsf.org/gnu/gnu-history.html>.

15. *See* Stallman, *Linux and the GNU Project*, *supra* note 11.

16. *See id.*

17. *See Appendix A in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION, *supra* note 3, at 221, 223-24 (debate between Andrew Tanenbaum and Linus Torvalds in comp.os.minix regarding Linux).

18. *See* Glyn Moody, *The Greatest OS That (N)ever Was* (last modified Aug. 1997) <http://www.wired.com/wired/5.08/linux.html>.

19. *See* Stallman, *Linux and the GNU Project*, *supra* note 11.

operating system grew and expanded through the open source processes to be discussed, eventually growing into a complete operating system package that is available, both freely, and as a part of a commercial distribution. Although estimates are difficult to pin down accurately, a good estimate for the number of current users of Linux is about 7.5 million.[20]

Linux is the open source project that has probably undergone the greatest number of publicized tests, reviews and comparisons. These have consistently shown Linux to be of superior technical quality, performing equal to, or better than, most of the products with which it is compared.[21]

4. Mozilla

Mozilla is the open source version of Netscape's Communicator. On January 23, 1998, Netscape announced that, in addition to giving away its Communicator product, its previously closed-

20. *See* Robert F. Young, *Sizing the Linux Market, Second Edition* (last modified Mar. 5, 1998) <http://www2.linuxjournal.com/enterprise/linuxmarket.html>.

21. *See* Eric Hammond, *1997 Product of the Year: Operating Systems - Network Operating Systems* (visited Jan. 27, 2000) <http://www.infoworld.com/cgi-bin/displayTC.pl?97poy.win3.htm#linux> (naming Red Hat Linux as the best network operating system of 1997); John Kirch, *Microsoft Windows NT Server 4.0 versus UNIX* (last modified Aug. 7, 1999) <http://www.unix-vs-nt.org/kirch> (comparing Windows NT to Linux and other UNIX operating systems, including a favorable direct technical and feature comparison of Linux with Windows NT); Henry Baltazar, *Linux: Enterprise Ready - The New Linux: 2.2.0 Kernel* PC WEEK ONLINE, Feb. 1, 1999 <http://www.zdnet.com/pcweek/stories/news/0,10228,387766,00.html> (technically reviewing Linux 2.2); Steven J. Vaughan-Nichols & Eric Carr, *Linux Up Close*: *Time To Switch*, SMART RESELLER, Jan. 25, 1999 <http://www.zdnet.com/sr/stories/issue/0,4537,387506,00.html> (technical comparison of versions of Linux); Quinn P. Coldiron, *Replacing Windows NT Server with Linux* (last modified Mar. 2, 1998) <http://citv.unl.edu/linux/LinuxPresentation.html> (analysis of trial of Linux use to replace Windows NT); Murry Shohat, *Engineers Speak Out: Linux vs. Windows NT, Part 1* (last modified Jul. 1998) <http://www.isdmag.com/Editorial/1998/CoverStory9807.html> (Review of engineer's responses to a request for evaluations of Linux). *But see, Linux: How Good Is It?* (abstract) (visited Feb. 6, 2000) <http://www.dhbrown.com/dhbrown/linux.cfm> (finding the enterprise capabilities of UNIX and Microsoft NT superior to Linux); *A File and Web Server Comparison: Microsoft Windows NT Server4.0 and Red Hat Linux 5.2 Upgraded to the Linux 2.2.2 Kernel* (last modified Apr. 13, 1999) <http://www.mindcraft.com/whitepapers/nts4rhlinux.pdf> (test sponsored by Microsoft which found that NT beat Linux in performance tests).

source Communicator software was going to become open source.[22] Not long after the source code was made available, a group of developers added 128-bit encryption capabilities, releasing a "Cryptozilla" product.[23] Netscape's official Communicator version 5.0 incorporated modifications suggested by open source contributors.[24]

5. Perl

Perl is a programming language that has become one of the most popular languages for Web page development, Internet services, graphical programming and many other purposes.[25] Perl "is the engine behind most of the 'live content' on the Web."[26]

6. Sendmail

Sendmail is a utility that routes about 80% of the e-mail on the Internet.[27]

B. *Initial Stages of Open Source Development*

Any open source project begins with the desire of a developer to meet some currently unfulfilled or inadequately fulfilled need. As one commentator artfully wrote, "[e]very good work of software starts by scratching a developer's personal itch."[28] To put the point more generally, this desire also accounts for programs that meet some need recognized by the programmer, even if that need

22. *See Our Mission* (last modified June 1, 1999) <http://www.mozilla.org/mission.html>.

23. *See* Michael Stutz, *Cryptozilla Thwarts Feds Crypto Ban* (last modified Apr. 3, 1998) <http://www.wired.com/news/news/technology/story/11465.html>.

24. *See Netscape's Brain Transplant* (last modified Nov. 10, 1998) <http://www.wired.com/news/news/technology/story/16163.html>.

25. *See What Is Perl?* THE PERL JOURNAL, THE VOICE OF THE PERL COMMUNITY (last modified Oct. 23, 1998) <http://tpj.com/whatisperl.html>.

26. *Open Source Products* (last modified Feb. 18, 1999) <http://www.opensource.org/products.html>.

27. *See* Josh McHugh, *For the Love of Hacking*, FORBES, Aug. 10, 1998 at 94.

28. Eric S. Raymond, *The Cathedral and the Bazaar: The Mail Must Get Through* (last modified Aug. 8, 1999) <http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar-2.html>.

is not one the programmer experiences personally.[29]

Further, "[g]ood programmers know what to write. Great ones know what to rewrite (and reuse)."[30] In the open source community, where the source code for existing programs is available, developers are more likely to reuse this code in developing new programs. When a substantial amount of open source software exists, there is an excellent chance of finding code that can be modified to meet a particular need. This eliminates the necessity of inefficiently reinventing the wheel.

Reusing existing code does not mean that open source software developers escape the trial and error process.[31] As Eric Raymond put it, "you often don't really understand the problem until after the first time you implement a solution. . . . So if you want to get it right, be ready to start over *at least* once."[32] This process can come in several forms. First, it may be the case that a new piece of software is in the process of being created using some code from an existing program when a program that would provide a better basis for modification is found.[33] Second, an existing program may provide an initial framework for development; a framework that is eventually replaced as the project progresses.[34] Finally, subparts of a larger project may have several prototypes developed, with only one (or parts of several) chosen for ultimate refinement through the open source process and inclusion in the larger project.[35]

29. *See* Richard Stallman, *The GNU Operating System and the Free Software Movement, in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION, *supra* note 3, at 53, 64 (discussing the fact that "many essential pieces of GNU software" were created as part of a plan to create a free operating system, rather than from a particular need for the software on the part of the programmer).

30. Raymond, *Mail Must Get Through*, *supra* note 28.

31. Eric S. Raymond's associated rule is "[p]lan to throw one away; you will, anyhow." *Id.* (citing Fred Brooks, THE MYTHICAL MAN-MONTH, Chapter 11).

32. *Id.*

33. This is what Eric S. Raymond describes happening with his efforts to modify fetchpop into a POP3 client for handling e-mail, which he eventually abandoned in favor of modifications to another program, popclient. *See id.*

34. This is what happened with Linux. Linus Torvalds started with Minix, a Unix-like operating system for 386 machines. Eventually all the original Minix code was replaced, but it had provided a basis upon which to found the initial efforts. *See id.*

35. Apparently, this occurred in the development of Linux. *See Halloween I: Open*

C. *Review by the Open Source Community*

There are several stages to the peer review process that are part of the success of the open source model. These stages include attaining technical prerequisites before soliciting feedback and developing a base of users/developers. This user base must then be encouraged to maintain an active role in the program's development. This is accomplished in part through the exercise of the traits necessary for project leadership.

1. Technical Prerequisites

Prior to submission to the open source community, a piece of code must attain a minimal level of technical development.[36] Once sufficient development has occurred, the software is sufficiently advanced to undergo community review. A second technical prerequisite is a means of handling communication with contributors. Contributors to an open source program are often geographically dispersed. For larger or more successful projects, the contributors may also be substantial in number. [37] Use of modern communication technologies - project web pages, mailing lists, newsgroups, et cetera - are necessary to facilitate the desired peer feedback that is at the heart of the open source process. In short, the growth of the Internet and Internet technologies has made the open source method possible.[38] Finally, the technology to engage in debugging and development must be available to the user for them to contribute. In the case of the Linux OS, for example, the act of installing

Source Software – A (New?) Development Methodology, (last modified Aug. 11, 1998) <http://www.opensource.org/halloween/halloween1.html>.

36. "It's fairly clear that one cannot code from the ground up in bazaar style. One can test, debug and improve in bazaar style, but it would be very hard to *originate* a project in bazaar mode. Linus didn't try it. I didn't either. Your nascent developer community needs to have something runnable and testable to play with." Eric S. Raymond, *The Cathedral and the Bazaar: Necessary Preconditions for the Bazaar Style* (last modified Nov. 20, 1998) <http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar-9.html>.

37. *See Halloween I*, *supra* note 35.

38. *See id.* Consequently, it is also reasonable to conclude that access to the Internet is necessary for participation in open source projects. Thus, Internet grow, not only in terms of technology, but also in terms of world penetration is important for open source software.

the debugging and development environment is implicit to the act of installing the operating system.[39] Thus, participation by Linux users in open source software development is facilitated by the common use of GNU tools for development used in the open source community.[40]

2. Obtaining Peer Review

Initially obtaining peer review is an important issue in itself. There are two principle ways to garner community involvement. The first is to take over ownership of a relevant existing program and make use of the existing user/developer base. The second possibility is to start a new project and solicit support from the open source community generally.

The easiest way to get peer review of a program is to take over an existing open source project which will serve as a basis for modification. The first way to do this "is to have ownership of the project handed to you by the previous owner (this is sometimes known as 'passing the baton')."[41] The owner of a project is seen as having a duty to pass on the ownership of a project that he or she no longer is able to or wishes to maintain.[42] Alternatively, ownership of an existing project may be obtained if the project needs work and the owner no longer maintains the project. In conformance with community norms, the would-be successor must first attempt to find the owner. Then it is appropriate to announce ownership of the project in as many relevant forums as possible, allowing substantial time to pass. If anyone claims to have been working on the project during this period of time, their claim will have priority. If no one makes such a claim, the successor may take over the project, and the existing user/developer base that accrues to the project's new owner.[43] Ownership of the project provides access to the existing userbase from whom it is possible to

39. *Id.*

40. *See id.*

41. Eric S. Raymond, *Homesteading the Noosphere: Ownership and Open Source*, (last modified Nov. 21, 1998)
<http://www.tuxedo.org/~esr/writings/homesteading/homesteading-4.html>.

42. *See* Raymond, *Mail Must Get Through*, *supra* note 28, ¶ 2.2.

43. *See* Raymond, *Ownership and Open Source*, *supra* note 41.

solicit feedback and development effort regarding any changes in the code.

The process of creating a new project can be difficult, particularly in the beginning. Brian Behlendorf, co-founder of the Apache Group, detailed the resource requirements necessary to start an open source project. There must be a project "captain" who oversees any changes to the code, fixes incompatibilities in contributions and in general "has overall responsibility for the quality of the implemented code."[44] Someone must perform "infrastructure support," maintaining mailing lists, the web server, bug database and other resources. In addition, there must be maintenance of the bug database - receiving reports about bugs, and responding to valid bug issues. The project must also be documented, which means not only providing explanations of the project and its current status, but also maintaining the Web site.[45] Someone also needs to "build momentum" for the project among other developers and users who will try the project and make peer-review contributions.[46] In addition to these roles, it is necessary to have people who can work on the development of the code until users join the project.[47]

3. Converting User Base Into Developers

Once there is a user base for a program, the open source process takes advantage of these users for program development.[48] Specifically, these users are sources, not only of feedback regard-

44. *See* Brian Behlendorf, *Open Source as a Business Strategy, in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION, *supra* note 3, at 149, 163.

45. *See id.* The existence of a Web site ("home" page) for the project has interesting implications for the "ownership" nature of an inherently abstract thing, like an open source project. A web page reinforces the idea of ownership as it relates to the more physical, territorial use of the term. *See* Eric S. Raymond, *Homesteading the Noosphere: Noospheric Property and the Ethology of Territory* (last modified Nov. 21, 1998) <http://www.tuxedo.org/~esr/writings/homesteading/homesteading-14.html>.

46. Behlendorf, *supra* note 44, at 164.

47. *Id.*

48. *See* Eric S. Raymond, *The Cathedral and the Bazaar: The Importance of Having Users*, (last modified Nov. 20, 1998) <http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar-3.html>; Eric S. Raymond, *The Cathedral and the Bazaar: Release Early, Release Often* (last modified Nov. 20, 1998) <http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar-4.html>.

ing flaws or shortcomings in the program, but sources of solutions for these problems. In highly planned, hierarchical development approaches, finding and fixing bugs is a long and arduous process, taking "months of scrutiny by a dedicated few."[49] This necessitates long periods of time between releases and disappointment when bugs remain in these long-awaited products. In contrast, bugs are relatively shallow, and easier to find and solve, in software subjected to the open source approach. Users of open source programs are encouraged to contribute not only identifications of bugs, but potential solutions as well. Thus, bugs are not only identified quickly, but given the abilities of the average open source user, a solution is quickly found.[50]

The ability to fix bugs quickly is analogized to the "Delphi effect" - the fact that the averaged opinion of a group of individuals with equal knowledge is much more reliable than the opinion of one randomly-chosen individual. The open source development process has shown "that the Delphi effect can tame development complexity even at the complexity level of an OS kernel."[51] While this is due in part to the fact that averaged opinions are more reliable, it is also due to the fact that, although open source software development "requires debuggers to communicate with some coordinating developer, it doesn't require significant coordination between debuggers."[52] This means that the intricacies and management costs associated with adding more developers to a hierarchical project are minimal in the distributed open source context.[53]

This difference from the structured, hierarchical model also allows much shorter release intervals. Releasing more often yields more corrections, and ultimately results in a high-quality piece of software developed relatively quickly.[54] This also helps minimize

49. Raymond, *Release Early, Release Often*, *supra* note 48.

50. Eric S. Raymond offers three alternative formulations of this point: (1) "Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone," (2) "Given enough eyeballs, all bugs are shallow," and (3) "Debugging is parallelizable." *Id*.

51. *Id.*

52. *Id*. (quoting Jeff Dutky).

53. *See id.*

54. *See id*.

the administrative costs associated with peer-reviewed open source methods. By releasing new versions often and incorporating bug fixes obtained through feedback, the duplicative efforts of debuggers are kept to a minimum.[55]

4. The Role of Leadership

Although the open source model involves much more distributed participation than the traditional, centralized, proprietary software development method, there are important roles for project leaders to play. In addition to handling the logistics of incoming bug reports and bug fixes from users, it is necessary to select among them. Only certain fixes can be implemented, and owners must select among the (potentially) many alternatives to find the solution that will ultimately be used.[56] However, there is more to the role than just recognizing good ideas. Often, different perspectives yield different characterizations or conceptions of a problem. These different perspectives can be insightful in determining how to fix problems.[57] Finally, suggestions for code changes may come, not only in the form of patches to fix problems, but code streamlining as well. As Eric Raymond noted, "[p]erfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away."[58]

A project owner must also resolve disputes arising in the project. The project head has the authority, under open source community norms, to make ultimate design decisions and help keep a group from breaking into multiple branches ("forking").[59] The issue of giving credit for contribution to a project is a more difficult issue to resolve, but also involves the project owner. Decision-making is easiest if the project is structured according to the "be-

55. *See id*.

56. *See* Eric S. Raymond, *The Cathedral and the Bazaar: Popclient becomes Fetchmail*, (last modified Nov. 20, 1998) <http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar-6.html>.

57. *See id.*

58. *Id*.

59. *See* Eric S. Raymond, *Homesteading the Noosphere: Causes of Conflict* (last modified Nov. 21, 1998) <http://www.tuxedo.org/~esr/writings/homesteading/homesteading-14.html>.

nevolent dictator" model.[60] This model of ownership consists of a single leader who makes design and group maintenance decisions, and also assigns project credit as constrained by community norms.[61] In the simplest sense, credit for contribution means ensuring that contributors receive a fair reputational stake in the success or failure of a project. As a project develops, tiers of contributors can arise, consisting of ordinary contributors and co-developers.[62] The co-developers get greater decision-making power over the conflicts formerly resolved solely by the "dictator." Even further removed from the benevolent dictator model are the leadership committee and rotating dictatorship models. These models involve turning co-developers into a leadership committee, or passing control among co-developers. These models are generally more complicated and less stable than the "benevolent dictator" model.[63]

Several character traits are important for project leaders as well. One important trait, even before substantial development begins, is strong people and communication skills.[64] These skills are helpful in attracting others to the project, and keeping developers happy so that they enjoy working (typically for free) on the project. Further, a somewhat humble, or at the very least non-egotistical, non-self-promoting, individual is necessary, given the community norms. This not only provides confidence in the project leader's ability to evaluate the work of contributors, but helps assure that participants are able to claim for themselves the prestige that they are entitled to - a critical element for participation.[65]

D. *Open Source Culture*

The discussion of "user-developers," taken alone, neglects an important characteristic of the open source community that facili-

60. *See* Eric S. Raymond, *Homesteading the Noosphere: Project Structures and Ownership* (last modified Nov. 21, 1998) <http://www.tuxedo.org/~esr/writings/homesteading/homesteading-16.html>.

61. *See id.*

62. *See id.*

63. *See id.*

64. Raymond, *Necessary Preconditions*, *supra* note 36.

65. *See infra* notes 77-81 and accompanying text.

tates participation. A user base must be converted from mere users to active contributors to the code development. Possibly the prime factor which leads to participation in the open source development process is the so-called "gift culture" factor.[66] "In gift cultures, social status is determined not by what you control but by what you give away."[67] There are several particular reasons why community reputation may lead to participation. The strongest reward is the pleasure of the good reputation itself.[68] Prestige within the open source community may allow a programmer to more readily persuade others to join projects owned by such a person, or to place particular value on that person's input.[69] To a lesser extent, the prestige of a developer in the open source community may also spill over to the exchange economy, placing them in higher demand within that market. While the gift-culture idea may be counter-intuitive to many businesses, it has been used to explain philanthropy and donation of resources in general.[70] Indeed, following from the analogies to philanthropy, it is reasonable to expect that open source participants' identities in their own eyes and the eyes of others may to come from their role in open source pro-

66. *See generally* Eric S. Raymond, *Homesteading the Noosphere*, (last modified Nov. 21, 1998)
<http://www.tuxedo.org/~esr/writings/homesteading/homesteading.html>.

67. *See* Eric S. Raymond, *Homesteading the Noosphere: The Hacker Milieu as Gift Culture* (last modified Nov. 21, 1998)
<http://www.tuxedo.org/~esr/writings/homesteading/homesteading-6.html>.

68. *See* Eric S. Raymond, *Homesteading the Noosphere: The Many Faces of Reputation* (last modified Nov. 21, 1998)
<http://www.tuxedo.org/~esr/writings/homesteading/homesteading-8.html>.

69. *See id.*

70. Susan Rose-Ackerman argues that "[o]ne explanation for giving is that donors benefit from the act of giving itself." *Altruism, Nonprofits, and Economic Theory*, 34 J. ECON. LIT. 701, 712 (1996). Thus, even though a person may benefit from someone else's donation to some charity, they may well prefer to donate the sum themselves because they value their own act of charity. *See id.* She notes that one benefit donors may get from their own charity comes from a "buying-in" mentality. "They may feel that they deserve to feel good about the charitable program only if they have made some marginal contribution to it," and further, "some charities are so small and some donors are so wealthy that individual gifts do affect service levels in observable ways." *Id*. at 713. This is consistent with the "homesteading" label on developer behavior in Eric S. Raymond's observations of the open source community. *See* Eric S. Raymond, *Homesteading the Noosphere: Locke and Land Title*, (last modified Jan. 1, 1999) <http://www.tuxedo.org/~esr/writings/homesteading-5.html>.

jects.[71]

Open source social ownership customs[72] provide a background in which esteem may be granted or withheld in a manner that supports the open source community. Ownership in the open source context means "hav[ing] the exclusive right, recognized by the community at large, to re-distribute modified versions [of a program]."[73] This idea is in tension with the ideology of the open source movement, as expressed in licensing terms; namely, the idea that the source code should be available to be freely modifiable by *anyone*.[74]

Another indication of why users may become developers comes from what draws many users to open source software in the first place. Specifically, the control over the code that open source software allows users has significant appeal. Thus, many of the users that are drawn to open source software are drawn by the prospect of being able to make their own changes to the code.[75]

71. For example, it was observed that among philanthropists in New York, "that involvement with particular organizations becomes part of donors' own identity in the eyes of those they know." FRANCIE OSTROWER, WHY THE WEALTHY GIVE: THE CULTURE OF ELITE PHILANTHROPY (1995). "[O]ne important implication is that individuals derive prestige from their identification with organizations and the elite networks with which they are associated." *Id.*

72. These are discussed in part, *supra* notes 41 - 47 and accompanying text.

73. Raymond, *Ownership and Open Source*, *supra* note 41.

74. *See infra* Section I.F (discussing open source licenses). However, this is not a direct conflict. Anyone can still make changes to their own copy. They have no guarantee of having it implemented in the main project, however.

75. "Subsequent improvements [to projects such as Apache] of the code often stem from individuals applying the code to their own scenarios." *Halloween I*, *supra* note 35. *See also* Dibona, *Introduction*, *supra* note 3, at 13-14 (noting that most open source projects begin when someone is looking for a tool to do a job and finds none, or one that is poorly maintained); Robert Young, *Giving it Away* in OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION 113, 117-120 (1999) (discussing the appeal provided by control over source code); Frank Hecker, *Setting Up Shop: The Business of Open Source Software*, (last modified Dec. 6, 1999) <http://www.hecker.org/writings/setting-up-shop.html> (noting a number of practical reasons why access to the source code is valuable to customers, including the ability to maintain their software even if the vendor goes out of business, the ability to fix bugs themselves if the vendor is unwilling to do so, or to port the software to platforms not otherwise supported by the vendor); Young, *Giving It Away*, *supra* at 120 (discussing NASA's choice of open source software due to their need to customize the code - they require a level of performance not available from any standard distribution of a program).

Coupled with this is the interaction between project owners and users, which can facilitate turning users into user-developers. Examples from open source projects indicate that a combination of encouraging participation among users, specifically soliciting comments regarding design decisions, implementing suggested changes and praising users when they provide patches and feedback, leads to further participation.[76] These can help encourage users, who may already be inclined to make improvements to the code, to resubmit these improvements to the project.

The norm against explicitly egotistical behavior,[77] which would appear to be inconsistent with the role esteem-seeking plays in the open source community, may actually help drive participants to a higher standard of contribution.[78] The norm helps assure that "one's work is one's statement."[79] This, in turn, ensures that the participants are driven toward a high level of performance, because rewards only come from a peer determination of program quality. Further, because code from self-promoting individuals is not rewarded, such "noise" is filtered out of the open source development discourse.[80] Finally, self aggrandizement is inconsistent with the quality of intelligent selection of code, necessary for a good

76. "If you treat your beta-testers as if they're your most valuable resource, they will respond by becoming your most valuable resource." Eric S. Raymond, *The Cathedral and the Bazaar: When Is A Rose Not A Rose?* (last modified Nov. 20, 1998) <http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar-5.html>. See also id. (discussing his efforts and their results regarding an open source software project); Marshall Kirk McKusick, *Twenty Years of Berkeley Unix, in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION, *supra* note 3, at 31, 42 (contributors were solicited to rewrite Unix utilities from scratch, compensated only by being listed among the contributors. Contribution began slowly, but as the list of contributors grew, the rate of contribution grew.) *Cf.* Jim Hamerly, et. al, *Freeing the Source, in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION, *supra* note 3, at 197, 201-02 (discussing the process as applied to the development of their open source license).

77. *See* Eric S. Raymond, *Homesteading the Noosphere: The Problem of Ego* (last modified Nov. 21, 1998)
<http://www.tuxedo.org/~esr/writings/homesteading/homesteading-10.html>.

78. However, Eric S. Raymond notes that this norm "has made it emotionally difficult for many hackers to consciously understand the social dynamics of their own culture[.]" *Id.*

79. Eric S. Raymond, *Homesteading the Noosphere: The Value of Humility* (last modified Nov. 21, 1998)
<http://www.tuxedo.org/~esr/writings/homesteading/homesteading-11.html>.

80. *See id.*

project leader. It is also inconsistent with the proper distribution of esteem by the leader to contributors, necessary for sustained user-developer contributions.[81] Thus, the norm against such behavior helps strengthen the quality of individual that ultimately leads an open source project.

The pure pleasure of hacking is another reason why individuals contribute to open source projects.[82] For some people, the enjoyment of programming can be satisfied through open source projects, which, in particular, may allow for greater creativity and experimentation than opportunities in the proprietary software world. A further aspect of this justification for participation comes from the enjoyment of creating a beautiful program, above and beyond any reputational benefits that may come from its creation.[83] However, this is intimately intertwined with the reputational benefits of the open source system. When contributions consist, not of entire programs, but of particular patches, project leadership, et cetera, it is difficult to evaluate the relative "beauty" of a particular developer's contribution. Thus, the knowledge that a contribution is technically superior comes from the critical review provided by the open source model.[84]

The norms of the open source community can also serve as a hurdle to participation. Such hurdles are of at least three different types. First, there are "password-like" mysteries. [85] Groups pre-

81. *See id.* Taking excess esteem for him or herself means that others in the project will be under-compensated in terms of esteem. Self-promotion may also lead others in the open source community to screen out the "noise" of that project, reducing the amount of esteem potentially available from the project.

82. *See* Eric S. Raymond, *Homesteading the Noosphere: The Joy of Hacking* (last modified Nov. 21, 1998)
<http://www.tuxedo.org/~esr/writings/homesteading/homesteading-7.html>.

83. *See id*. *See also* DiBona, *supra* note 3 at 13 ("Much like the rush a runner feels while running a race, a true programmer will feel this same rush after writing a perfect routine or tight piece of code. It is difficult to describe the joy felt after completing or debugging a hideously tricky piece of recursive code that has been a source of trouble for days.")

84. *See* Raymond, *The Many Faces of Reputation*, *supra* note 68.

85. Eric S. Raymond, *Homesteading the Noosphere: Acculturation Mechanisms and the Link to Academia* (last modified Nov. 21, 1998)
<http://www.tuxedo.org/~esr/writings/homesteading/homesteading-18.html>.

vent participation by anyone who has not uncovered the mystery.[86] Second, there is often a requirement of understanding of some particular technical issue. This serves as a proxy for the participant's overall technical ability to contribute to the project.[87] Finally, through the examples of other hackers working on a project, a new participant is expected to learn both the procedural and social rules and norms that govern behavior. [88] Such norms play an important role in the control of the open source community by aiding in the acculturation of new members. However, it is important to note that barriers to participation which are not associated with the goals of acculturation or ensuring technical proficiency could have negative consequences.[89]

E. *Modularity*

Modularity[90] of code plays an important role in open source development as well. The advantages of modular code include:

1. If a function performed by a module changes, only that module changes and the rest of the program is unaffected.

2. If a new program feature is added, a new module or hierarchy of modules to perform that feature can be added.

3. Program testing and retesting is easier.

86. "As one example, there is a USENET newsgroup called alt.sysadmin.recovery that has a very explicit such secret; you cannot post without knowing it, and knowing it is considered evidence you are fit to post. The regulars have a strong taboo against revealing this secret." *Id*.

87. *See id.*

88. *See id*.

89. For example, it has been suggested that the reason there was forking, a rare occurrence in open source projects, in the case of BSD Unix was due to the fact that not everyone can contribute to the BSD codebase. Thus, forking could be driven by the hope of establishing a project that could supplant the existing project in popularity and acceptance. *See Halloween I*, *supra* note 35. This is not likely to be a problem with the existing barriers - the excluded members have qualities (failure to follow norms, low technical ability) that makes them unlikely to have the ability to start a project that can truly compete.

90. A modularized program involves "constructing a program as a set of conceptually and operationally independent pieces (modules)." JAMES MARTIN & CARMA MCCLURE, SOFTWARE MAINTENANCE: THE PROBLEM AND ITS SOLUTIONS 79 (1983).

4. Program errors are easier to locate and correct.

5. Program efficiency is easier to improve.[91]

Thus, modularity can make project leadership a manageable task by making the program easier to maintain.[92] It allows projects to be divided up into discreet tasks, with programmers working in parallel, yet not creating conflicting changes.[93] Modularity may also facilitate the borrowing of portions of code, making the code for particular tasks easier to find in old programs.[94] The program design necessary to allow the insertion of code borrowed for a discreet task will also encourage modular design in the program being created. The discreet, understandable nature of modular code can facilitate peer evaluation. Finally, "[t]he traditional approach for enhancing program quality is modularization."[95]

F. *Open Source Licenses*

"Open Source lives or dies on copyright law."[96] The licenses applied to open source software play an important role in whether future versions or changes to the software remain open source. Indeed, the licensing terms of software are critical to the determination of whether it meets the formal "Open Source" definition.[97]

91. *Id*. at 79-80.

92. *See id.* at 79. For example, Linus Torvalds stated that, in leading a project, "[w]ithout modularity I would have to check every file that changed, which would be a lot, to make sure nothing was changed that would effect anything else." Torvalds, *The Linux Edge*, *supra* note 12 at 108. On the other hand, "[w]ith modularity, when someone sends me patches to do a new filesystem and I don't necessarily trust the patches *per se*, I can still trust the fact that if nobody's using this filesystem, it's not going to impact anything else." *Id*.

93. *See id.*

94. *See* Mark A. Lemley & David W. O'Brien, *Encouraging Software Reuse*, 49 STAN. L. REV. 255 (1997) (discussing value of modularity to software reuse).

95. MARTIN, *supra* note 90, at 79.

96. Larry Wall, *Diligence, Patience, and Humility*, *in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION, *supra* note 3, at 127, 142.

97. As a way of reliably knowing whether a piece of software is open source, the Open Source Initiative ("OSI") has registered the term "OSI Certified" as a certification mark. *See The OSI Certification Mark and Program* (visited May 18, 2000) <http://www.opensource.org/certification-mark.html>. Using the term "OSI Certified" requires compliance with the Open Source Definition. *Id.* The Open Source definition requires that a license "not [to] restrict any party from selling or giving away the software

There are a number of different types of licenses that are consistent with the requirements of the open source definition.[98]

1. GNU General Public License (GPL)

The GNU GPL is the most well-known and widely used of the open source licenses. An important initial note about the GNU GPL is that it contains not only the licensing terms, but a discussion of the justifications for those terms.[99] The basic requirements of the GPL are that "enhancements, derivatives, and even code that incorporates GPL'd code are also themselves released as source code under the GPL."[100] Thus, modifications to GPL software cannot be made closed-source, and no GPL program can be incorporated into a proprietary program.[101]

2. The GNU Library GPL (LGPL)

The LGPL is derived from the GPL for use with software libraries. The primary difference is that "a LGPL-ed program can be incorporated into a proprietary program."[102] GNU is currently

as a component of an aggregate software distribution containing programs from several different sources. The license may not require a royalty or other fee for such sale." *The Open Source Definition* (last modified Dec. 20, 1998) <http://www.opensource.org/osd.html>. "The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software." *Id.* "The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties." *Id.* Further, the license may not discriminate against persons or groups or fields of endeavor. *See id.* These are just some examples of the license requirements. For others, *see id.*

98. These licenses are intended to cover the most widely-used licenses, and is not intended to be an exhaustive list of licenses that comply with the open source definition.

99. *See The General Public License (GPL)* (last modified June 1991) <http://www.opensource.org/licenses/gpl-license.html>.

100. Behlendorf, *supra* note 44, at 167. *See also The General Public License (GPL)*, *supra* note 99. Behlendorf notes that this aspect of the GPL is "viral" in nature, in that "there is no chance of a commercial interest forking their own development version from the available code," creating a closed-source product that they can then market and sell exclusively. *Id.*

101. *See* Bruce Perens, *The Open Source Definition*, *in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION, *supra* note 3, at 171, 182. For purposes of the GNU GPL the term "proprietary" means "any program with a license that doesn't give you as many rights as the GPL." *Id.*

102. *Id*

discouraging the use of the LGPL, in favor of the GPL instead.[103]

3. The BSD-style License

Another popular license, the BSD-type license, is used by Apache and BSD-based Unix operating systems. This license has been summed up as stating "'[h]ere's this code, do what you like with it, we don't care, just give us credit if you try and sell it.'"[104] This does allow incorporation of the code in proprietary products, which is seen by some as an advantage for "common" protocols or services.[105] However, there are also risks because "no incentive is built into the license to encourage companies to contribute their code enhancements back to the project.[106] The apparently benign requirement that advertising of products including BSD-licensed software must give credit to Berkeley University can become untenable in a large, multi-component distribution, which could require pages of footnotes.[107]

103. *See* Richard Stallman, *Why You Shouldn't Use the Library GPL For Your Next Library* (last modified Nov. 6, 1999) <http://www.fsf.org/philosophy/why-not-lgpl.html>. The original GNU C library was issued under this, because the number of existing C libraries meant that restricting use to GPL developers would merely have led proprietary developers to use another C library. With new libraries, the theory was that restricting them to GPL developers could give them a competitive advantage. *See id.*

104. *See* Behlendorf, *supra* note 44, at 164. For examples of the BSD-style license, *see The BSD License* (last modified Nov. 30, 1998) < http://www.opensource.org/bsd-license.html>; *The Apache License* (last modified Feb. 16, 1999) <http://www.apache.org/LICENSE.txt>.

105. Apache chose the BSD-style license so that HTTP would become a true standard. It was not considered a problem if Microsoft chose to incorporate their HTTP engine into their products, since that was assumed to help keep HTTP a common protocol. *See* Behlendorf, *supra* note 44, at 165.

106. *Id.* This could be particularly problematic, give Microsoft's proposed strategy for dealing with open source software—namely, to "de-commoditize protocols & applications" *See Halloween I*, *supra* note 35. Under the BSD-style license it seems clear that Microsoft could incorporate open source standard protocols into their products, but make proprietary modifications, that would then propagate out through Microsoft's large OS user base, becoming the de-facto standard; a standard over which Microsoft has exclusive control.

107. "[T]he Debian GNU/Linux distribution contains over 2,500 software packages, and if even a fraction of them were BSD-licensed," this would require a substantial list of software and credit in an advertisement. Perens, *supra* note 101, at 183.

4. Mozilla Public License (MPL)

The MPL was the result of dissatisfaction with existing licenses for the particular needs of Netscape and Netscape Communicator. Its development resulted from a consideration of the advantages and benefits of existing license types, comment from the open source community and work by teams of Netscape employees.[108] Any changes to an MPL distribution must be released under the same copyright as the MPL, making it available back to the project. However, "'distribution' is defined as the files as distributed in the source code," meaning that the files could be incorporated into another, proprietary program.[109] This license also requires that anyone "contributing code back to the project release any and all claims to patent rights that may be exposed by the code."[110]

5. Netscape Public License (NPL)

The NPL is a Netscape-specific version of the MPL. It grants special privileges to Netscape that do not apply to anyone else. Essentially, it allows Netscape "to take [open source] modifications private, improve them, and refuse to give you the result."[111]

6. Artistic License

The Artistic license was originally developed for Perl, but is currently disfavored as compared to other licenses such as the GPL. The license "prohibits sale of the software, yet allows an aggregate software distribution of more than one program to be sold.[112] The Artistic license also requires modifications to be made open source, but then provides loopholes for taking releases private, or putting it in the public domain.[113]

108. *See* Hamerly, *supra* note 76, at 200-03.

109. Behlendorf, *supra* note 44, at 166.

110. *Id.* at 166-67. *See also*, *Mozilla Public License* (last modified Dec. 8, 1998) <http://www.mozilla.org/NPL/MPL-1.0.html>.

111. Perens, *supra* note 101, at 184. *See also Netscape Public License* (last modified Dec. 8, 1998) <http://www.mozilla.org/NPL/NPL-1.0.html>.

112. Perens, *supra* note 101, at 183-84. This loophole is substantial. By aggregating software under the Artistic license with a trivial program it can be sold as a "bundle."

113. *See id.*

G. *Important Pre-existing Facilitators of the Open Source Method*

There are a number of factors that made the open source development model possible that are not, strictly speaking, part of the development model itself. However, given the important roles these factors play as preconditions of open source success, they are worth consideration.

1. Academia

Given the important role that the culture and norms of the open source community play in the success of open source software, it is important to recognize the role that academia plays in teaching those norms. Richard Stallman, arguably the founder of the modern free software[114] movement, first experienced this type of community at The Massachusetts Institute of Technology in the 1970s. At the MIT Artificial Intelligence Lab there was a software-sharing community, allowing free use of software and free access to source code for anyone who wanted to use it. Although this culture changed due to commercial influences in the 1980s, the principles Stallman took away from the experience led him to recreate this type of community through the GNU project.[115] Thus, the free software movement has its roots in the source-sharing culture of the university computer science community. Similarly, the norms and culture of the university setting at large may themselves represent analogous cultural systems to the open source community. The activities of tenured professors, who no longer have concerns about "survival issues," become focused on "reputation enhancement" through intellectual achievement.[116] This is similar to the behavior of hackers in the open source gift culture.

The university setting continues to be not only a forum for in-

114. "Free software" is essentially the same as open source software, but is the preferred term of Richard Stallman. For uniformity, this paper has used "open source" throughout, but will use the term "free software" here to respect Stallman's preference for the term free software, for its implicit reference to principles of freedom.

115. *See* Stallman, *supra* note 29, at 53-58.

116. *See* Raymond, *Acculturation Mechanisms and the Link to Academia*, *supra* note 85.

troduction into the norms of open source society, but also a forum for introduction to open source technology. For example, much work on components of Linux and GNU was done by individuals at educational institutions. Open source software is frequently utilized by universities for purposes of computer science education, due in large part to the accessibility of the source code. Use of open source software in the university setting also means that new research ideas are often tried out first in open source software. Finally, universities may facilitate the distribution of open source programs to areas with only marginal Internet penetration.[117]

2. Communications Technologies

The Internet is a technology whose existence was critical for the development of the open source movement. "Open Source has been born into a digital renaissance made possible by the Internet, just as modern science was made possible during the Renaissance by the invention of the printing press."[118] The economic and physical barriers to software and source code distribution have been lowered by the Internet.[119] The Internet also brought together hackers in a community, rather than leaving them isolated in small groups.[120] Finally, the computers and the Internet allow the marginal cost of distributing software or source code to be zero.

3. Standards

Technical standards have played an important role in the ability of open source projects to go forward. Indeed, it is the elimination of open source community access to standards that Microsoft has raised as a primary means of preventing competition from Linux.[121] "OSS projects have been able to gain a foothold in many server applications because of the wide utility of highly commodi-

117. *See Halloween I*, *supra* note 35.

118. *See* DiBona, *supra* note 3, at 16.

119. *See id*.

120. *See* Eric S. Raymond, *A Brief History of Hackerdom*, *in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION, *supra* note 3, at 19, 20 (specifically, discussing the role of ARPANET).

121. *See Halloween I*, *supra* note 35.

tized, simple protocols."[122]

H. *The Business of Open Source Software*

Despite the feeling among some in the community that software should be given away, not sold,[123] the "business" of open source software dates back to its origins.[124] For the purposes of this discussion, the business of open source will include efforts to sell open source software, whether done for profit or merely to recover some costs. This business includes both the marketing and sale of pre-existing open source projects, or the "taking open source" of previously proprietary commercial products.

A number of companies have successfully based their business on selling open source software.[125] Part of what is being sold is simply a convenient aggregation of open source programs. However, many of these companies also provide value beyond this convenient aggregation. For example, traditional models of customer support[126] may be provided for the programs included in the

122. *Id.*

123. "Many people believe that the spirit of the GNU project is that you should not charge money for distributing copies of software, or that you should charge as little as possible—just enough to cover the cost." *See* Richard Stallman, *Selling Free Software*, (last modified Dec. 17, 1998) <http://www.fsf.org/philosophy/selling.html>. However, Stallman goes on to note that GNU "encourage[s] people who redistribute free software to charge as much as they wish or can." *Id.*

124. One of the first projects that originated what has now become the open source, or free software, movement was GNU Emacs. Richard Stallman made this available for free via ftp, but also would mail a copy on tape for a fee of $150. *See* Stallman, *The GNU Operating System*, *supra* note 29, at 58.

125. For example, Red Hat (www.redhat.com), Debian (www.debian.org) and S.u.S.E. (www.suse.com) sell versions of the Linux operating system. The basic economics of the market for proprietary software taken open-source, as well as the marketing points raised in the context of existing open source projects apply to these programs with equal force. Thus, the two ways of getting to an open source business will not be considered separately. However, one additional factor raised to justify changing proprietary programs to open source. In areas where there is a "commercial wall" between two open source programs, there will be a strong tendency for an open source program to arise to "bridge the gap." Behlendorf, *supra* note 44, at 160. It may be beneficial for the company to take the program open source preemptively. *See id.* The business could then use their existing brand reputation to continue to profit from the sale of a particular product even after it has become open.

126. What is meant by customer support here is what is called "hand-holding"—providing training and basic knowledge that helps the customer understand how to use

software package. These businesses may also help to implement changes suggested by their customers for immediate use by the customer in their environment, and perhaps also in future versions of the product. Because it is software whose source code is freely available that is being sold, the open source market is a commodity market.[127] Brand equity, therefore, is a selling point to a much greater extent than the underlying technology.[128]

Many in the open source community would most naturally point to the reliability and technical superiority of open source software as the primary marketing points.[129] However, business experience has taught that, while technical excellence is necessary for an open source business's success, it is not sufficient.[130] Rather, other points must be emphasized in addition to the technical merits in order to be successful. Open source software can be a means of lowering overhead.[131] Open Source software also may allow the support of a broader range of platforms than would be possible with proprietary software. For example, privately under-

the existing product. Customer service could also include responding to customer complaints and recommendations for changes or needed features for the software. *See* Young, *supra* note 75, at 115-17.

127. *See id.*

128. This point is emphatically made by Robert Young. He likens the marketing of Red Hat to the marketing of cars or ketchup. Even though ketchup can be made at home, the convenience of having Heinz or Hunts do it encourages purchases. However, it is the brand marketing of Heinz, rather than the technical superiority of their ketchup, that leads them to have 80% market share. *See id. See also*, Nicholas Petreley, *Linux and the Monopoly Game* (visited Feb. 10, 2000) <http://www.linuxworld.com/linuxworld/lw-1999-01/lw-01-penguin.html>.

129. *See, e.g.*, *The Business Case for Open Source* (last modified Dec. 18, 1998) <http://www.opensource.org/for-suits.html> (noting reliability and technical arguments); Michael Tiemann, *Future of Cygnus Solutions* in OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION, *supra* note 3, at 71, 83 (noting his initial marketing focus on the technical merits of GNU tools).

130. *See* Young, *supra* note 75, at 120 (stating that "[t]he benefit to using Linux is not the high reliability, ease of use, robustness, or the tools included with the Linux OS," but rather other features of the software).

131. Cygnus took the marketing approach of "explain[ing] to customers why they should buy from us instead of trying to do the work with their own people," noting that "their engineers would benefit from having us do the baseline porting, support, and maintenance work." Tiemann, *supra* note 129, at 83. This benefit will also be of importance in the decision to take proprietary software open source. *See also The Business Case for Open Source*, *supra* note 129.

taken ports of a program to a new platform will be contributed back to the project, allowing for incorporation into the next version of the product.[132] Because open source software is a commodity market, competitors may try to offer new features for the software. However, these features can simply be added into future versions of the general program that will be sold by all the business's competitors.[133] Thus, as long as a dominant company maintains its own high levels of performance, it is unlikely a competitor will be able to beat them merely by offering new software features.[134] Further, open source software allows a business to maintain close relations with customers, even to the point of "co-opting your customers' engineers to help your development."[135] This further lowers overhead, but also, by incorporating customer feedback and fixes into rapidly re-released products, allows heightened responsiveness to customer needs.[136] Implicit in the "co-opting" of a customer's engineers is the idea that customers can make their own modifications when needed. The ability to make needed or desired modifications to a program on their own is an extremely important selling point for many customers.[137]

I. *Freedom and Free Software*

An important part of the discussion surrounding the open source movement involves the issue of freedom, particularly as it

132. *See* Tiemann, *supra* note 129, at 83.

133. *See id.*

134. "Unlike proprietary software in which competitors fight in a two-sided win/lose contest, with Open Source it's more like fighting on a Moebius strip, and everything flows to the side of the primary maintainer." *Id* at 84. Thus, while competitors can easily enter the market because the product is freely available, the grounds on which existing businesses can be outperformed is limited in the long term.

135. *The Business Case for Open Source*, *supra* note 129.

136. *See* Tim O'Reilly, *Hardware, Software and Infoware*, *in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION, *supra* note 3, at 189, 195.

137. For example, NASA chose Red Hat Linux because the source code was available. Their performance standards were extremely exacting, and thus they needed to be able to modify the code to fit particular needs. *See* Young, *supra* note 75, at 120. Young goes on to state that "[t]he benefit to using Linux is not the high reliability, ease of use, robustness, or the tools included with the Linux OS. It is the benefit of *control* that results form the two distinctive features of this OS; namely, that it ships with complete source code, and that you can use this source code for whatever you chose—without so much as asking our permission." *Id.* at 120; *See also* Tiemann, *supra* note 129, at 86.

applies to software and intellectual property. The Free Software Foundation, and Richard Stallman in particular, have been the leading proponents of the freedom-enhancing aspects of "copylefted"[138] software. There are three specific freedoms associated with free software. "First, the freedom to copy the program and give it away to your friends and co-workers; second, the freedom to change the program as you wish, by having full access to source code; third, the freedom to distribute an improved version and thus help build the community."[139] Free Software supporters argue that "efforts to attract new users into [the free software] community are far outstripping the efforts to teach them the civics of our community."[140] Thus, the Free Software supporters argue against the use of the term "open source," as diverting the focus of attention away from freedom and "to appeal to executives and business users."[141]

II. INTRODUCTION TO COMPLEXITY THEORY

Complexity theory, as a subject matter, has its roots in many different fields of study. Aspects of the theory were developed in such diverse fields as genetics and economics to explain empirical observations that were not predictable by traditional theories. Specifically, traditional "linear" models were often unable to account for certain observed behaviors that seemed "nonlinear" in nature. Briefly, complex systems are groups of agents whose nature and behavior are governed by certain sets of rules. The nature and behavior of these agents lead to outcomes within the system and capabilities of the system making it greater than the sum of its parts. As will be seen, the complex nature of a system may be valuable in many ways, but also makes the prediction of specific future characteristics of the system difficult or impossible. Finally, it is important to note that complex systems may exist side-by-side with

138. Copyleft refers to the GNU-style license. Rather than withholding the ability to use the work, as with copyright, the ability to use the work is specifically granted under copyleft.

139. Stallman, *Overview of the GNU Project*, *supra* note 14.

140. Stallman, *The GNU Operating System*, *supra* note 29, at 69.

141. *Id.*

other types of systems.[142]

The discovery that a system is complex in nature could lead to two different conclusions for anyone concerned with the future of the system. First, attempts could be made to force the system to behave more linearly, resulting in a loss of the benefits of complexity, but perhaps a gain in predictability. Alternatively, the ambiguity in specific future states of the system could be accepted, with faith placed in the ultimate benefits that accrue to complex systems.

Greater explanation of the nature of complex systems has been detailed elsewhere.[143] Thus, a description of what constitutes a complex system will be outlined here in brief. The factors that make up a complex system are: (1) a systems of agents with certain internal traits, (2) interactions among these agents that occur following certain rules, and (3) the general consequences that can be expected to result.

A. *System of agents*

The term "agent" likely conjures up different, but nonetheless clear, pictures in the minds of everyone. In the law, an agent is "a person authorized by another to act on his account and under his control."[144] However, in a complex system an agent is merely any actor (be it a person, a computer program, or a gene) that has certain internal and behavioral characteristics. In complexity theory, the internal traits of an agent are considered to be: "(1) a performance system, (2) a credit-assignment algorithm, and (3) a rule-discovery algorithm," and, as will also be seen, (4) a mechanism for making predictions.[145]

142. *See* M. MITCHELL WALDROP, COMPLEXITY: THE EMERGING SCIENCE AT THE EDGE OF ORDER AND CHAOS 43 (1992).

143. *See id.*; *see also* PETER COVENEY & ROGER HIGHFIELD, FRONTIERS OF COMPLEXITY: THE SEARCH FOR ORDER IN A CHAOTIC WORLD (1995); JOHN H. HOLLAND, HIDDEN ORDER: HOW ADAPTATION BUILDS COMPLEXITY (1995); STUART KAUFFMAN, AT HOME IN THE UNIVERSE 19 (1995). For discussions of complexity geared toward the legal audience, *see* J.B. Ruhl, *Complexity Theory as a Paradigm for the Dynamical Law-and-Society System: A Wake-Up Call for Legal Reductionism and the Modern Administrative State*, 45 DUKE L. J. 849 (1996).

144. RESTATEMENT (SECOND) OF AGENCY, § 1, comment e.

145. HOLLAND, *supra* note 143, at 87.

"The performance system specifies the agent's capabilities at a fixed point in time—what it could do in the absence of further adaptation."[146] Basically, this includes the agent's ability to obtain information from its environment, as well as its ability to act on its environment on the basis of this information.[147] Implicit in this ability is the necessity of a "processing mechanism." This can be analogized to a set of if-then rules that allow the agent to determine what action is appropriate for the given bit of information with which it is dealing.[148]

The agent's credit assignment mechanism is a tool for evaluating the processing rules that it uses. This mechanism must somehow determine which if-then rules lead to good outcomes for the agent, and which do not. Competition between rules is used to find and reinforce rules with successful outcomes, and weed out those that are unsuccessful.[149] This is a challenging proposition for several reasons. First, the ease with which a rule is able to be evaluated will depend upon the role the rule plays. A rule that calls for direct interaction with the agent's environment will be easier to evaluate, because it will generate direct feedback from the environment. However, "[c]redit assignment is much more difficult when some early stage-setting action makes possible a later useful outcome."[150] Second, the credit assignment mechanism is dependent on the current status of the agent in its environment. Thus, the mechanism must be able to respond to changes in the agent and the agent's environment.[151]

A supplement to the credit assignment mechanism is the rule discovery mechanism. This process allows new if-then rules to be put into circulation for evaluation by the credit assignment process.

146. *Id.* at 88.

147. *See id.*

148. *See id.*

149. *See id.* at 89.

150. HOLLAND, *supra* note 143, at 53. Such a rule would be "if hungry, then look for area with characteristic X," would be harder to evaluate, because it does not (necessarily) result in helpful feedback from the environment. However, if areas with characteristic X are a likely source of food, the existence of the initial rule would allow the subsequent invocation of the rule "if you have food, then eat food." Thus, the mechanism must have some way of rewarding stage-setting rules.

151. *See id.*

This occurs principally by replacing the unsuccessful rules, which were weeded out by the credit mechanism, with new rules based on permutations of successful rules.[152] The two processes by which these permutations of successful rules are created are combination[153] and mutation.[154] Combination simply involves the inclusion of various elements of successful rules to create a new rule. Mutation is the alteration of a successful rule to create a slightly different rule.

Finally, agents need a prediction mechanism to allow the environmental feedback obtained by the agent to be translated into a broader set of rules than would be possible from the limited experiences of any agent. Agents with a means of making predictions are able to take the "building blocks" from their actual experiences, and by emphasizing patterns, use these blocks to create tools for dealing with patterns in their environment.[155] The best example of this is the human ability to reuse basic "building blocks" such as "tree," "car," or "person," to comprehend vast numbers of novel situations. Similarly, the laws of physics are relatively simple, and small in number, compared to the behavior of the world that they are able to describe.[156] "[I]f [you] have a process that can discover building blocks . . . the combinatorics start working *for* [you], rather than against [you]. [You] can describe a great many complicated things with relatively few building blocks."[157]

152. *See* HOLLAND, *supra* note 143, at 90.

153. This is commonly referred to in the literature as "sexual reproduction," but the term "combination" will also be used.

154. "Chemistry" is a related concept that applies to a wide variety of complex systems:

> The first source of chemistry's power is simple variety - unlike quarks, which can only combine to make protons and neutrons in groups of three, atoms can be arranged and rearranged to form a huge number of structures. The second source of power is reactivity: structure A can manipulate structure B to form something new - structure C.

WALDROP, *supra* note 142, at 314.

155. *See* HOLLAND, *supra* note 143, at 34-37.

156. *See id.* at 37.

157. *See* WALDROP, *supra* note 142, at 170 (quoting John Holland).

B. *Interactions of agents*

The interaction among agents leads to many of the beneficial aspects of complex systems. There are basic principles that describe the processes of interaction among agents in complex system, including the presence of flows and the use of tagging mechanisms. Agent interactions lead to changes in the system, not only in the co-evolution of agents and their environment, but through the multiplication and recycling of resources and diversity in the system. Finally, the interaction of agents allows them to form "meta-agents" which act as agents at a higher level.

1. Rules for Interaction

As John Holland observed, "complex large-scale behaviors [emerge] from the aggregate interactions of less complex agents."[158] Agent interaction is characterized by the use of tags and the existence of flows over a network of agents. Tagging allows agents to determine traits of their environment and other agents. Agents compare their own tags with the tags of other agents to determine if resources can be exchanged, and in what quantity. [159] Over time, in a complex system, this allows agents to specialize and cooperate.[160] Further, tags evolve in a manner similar to that of internal rules. As agent interactions occur, the system selects "*for* tags that mediate useful interactions and *against* tags that cause malfunctions."[161] Cooperation, specialization and tag evolution facilitates the emergence of meta-agents and general system characteristics that endure despite the continual evolution of the internal components of the system.[162]

In complex systems, agents can be considered "nodes" and the possible interactions of agents, defined by the tags, are "connectors" in this network.[163] As interactions occur between the nodes, "flows" of resources occur along the various connectors. These

158. HOLLAND, *supra* note 143, at 11.
159. *See id.* at 103-04.
160. *See id.* at 15.
161. *Id.* at 23.
162. *See id.* at 14-15.
163. *See id.* at 23

flows vary over time as the agents and their environments co-evolve based on their experiences with one another.[164] The importance of these flows will be seen to arise from the role they play in the changes they bring about in the system. The future of agents in a complex system are influenced by prior interactions of among agents and their environment.[165]

2. Interactions Yield Systemic Changes

The nature of the interaction of agents in a complex system leads to several large-scale results for the system itself. These results include system-wide changes from changes in flows, and diversity that results from the co-evolution of agents and their environment. The nature of flows in a complex system are such that the input of a resource at one node will result in that resource being spread throughout the system producing a chain of changes.[166] The fact that, as a result of flows, changes in the entire system can occur through the introduction of a new resource or the re-routing of an existing resource makes it nearly impossible to make long-range predictions based on simple trends.[167] A further characteristic of flows is that resources are recycled as they flow between nodes. This allows complex systems to be more productive for a given initial input of some resource than it would be if recycling did not occur.[168]

Changes in a complex system due to flows, the cooperation and

164. *See id.*

165. ROBERT JERVIS, SYSTEM EFFECTS 29-32 (1997).

166. For example, the purchaser of a home pays a contractor, who pays a tradesman, who in turn buys food, et cetera. At each stage, some of the money is saved and the rest is used for expenses. The effect of the initial contract is multiplied when its total effect is traced through the network. *See* HOLLAND, *supra* note 143, at 23-25.

167. *See id.* at 25.

168. John Holland uses the example of a three node system that turns steel into automobiles. One node supplies the ore, one node processes the ore into steel, and one node turns the steel into cars. He assumes one unit of ore produces one unit of steel which produces one unit of car. Finally, he assumes the steel producer sends half its output to the car manufacturer. If the cars are driven until they are unusable, then an input of 1000 units of ore will result in 500 units of cars. However, if 3/4 of the steel from cars is recycled, the same input of 1000 units of ore from the ore producer, when added with the units of steel recycled, will allow 800 cars to be produced. Thus, recycling can substantially increase the resources at each node. *See id.* at 25-26.

specialization that occur from tagging, and the co-evolution of agents and their environment result in substantial diversity in complex systems. If an agent is removed from the system, "the system typically responds with a cascade of adaptations resulting in a new agent that 'fills the hole.'"[169] A new agent or agents will arise to fill the need left by the removal of the agent, but may perform the tasks in different ways. These new agents may create new opportunities, or niches, to be exploited by other agents.[170] Thus, the minor change of removing a single agent from a system can result in the creation of new agents and changes in roles for existing agents, leading to greater diversity. This diversity is itself dynamic. When the diversity of a complex system is disturbed, it eventually settles back down to a pattern. However, in complex systems, this new pattern of diversity may be different than the old one. This also leads to further interactions and new niches. Thus, the diversity in a complex system is due both to adaptation to individual internal changes, as well as changes in the system as a whole.[171]

This co-evolution of agents occurs within a range of almost limitless possibilities[172] and a constantly co-evolving environment. As a result, there is no practical way of "optimizing" a given agent's fitness. Changes only occur based on the current abilities and surroundings of an agent - in particular those surroundings which the agent is able to perceive. This limited range of alternatives may not include the alternative that is "optimal."[173] Thus, agents attempt to maximize their "fitness" within the range of alternatives (the "fitness landscape") that are available to them at a given time.

169. *Id.* at 27.

170. *See id.* at 27-28.

171. *See id.*at 29-30.

172. This is sometimes referred to as a "fitness landscape." This landscape consists of "hills" of higher levels of fitness, and "valleys" of lower levels of fitness. *See* KAUFFMAN, *supra* note 143, at 161.

173. *See* WALDROP, *supra* note 142, at 151. An analogy would be trying to predict every possible move that could be achieved in chess, and from that determine what the "optimal" course of action. In reality, one can only predict a limited number of moves in advance, based upon the particular moves of a particular opponent. *See id.*

3. Interactions Allow the Formation of Meta-Agents

The cooperation and competition of agents allows the formation of "meta-agents," or aggregates of agents.[174] These meta-agents act as agents at a higher level. [175] The fact of agent aggregation is basic to all complex systems. The resulting features of the meta-agents, which arise from the historical interactions of lower-level agents, "are the most enigmatic aspect of [complex systems]."[176] "The behavior of these clusters, or meta-agents, is governed by the same principles that govern the underlying agents that aggregated in the first instance. This process of aggregation and re-aggregation often repeats numerous, yielding the hierarchical organization typical of complex systems."[177]

The behavior of these meta-agents is different than merely the sum of the capabilities of the constituent agents.[178] "[I]t is difficult to evolve a single agent with the aggregate's capabilities. Such complex capabilities are more easily approached step by step, using a distributed system."[179]

C. *Only Short-term Predictions*

Although it should be obvious from the discussion of complex systems thus far, it is worth stating clearly that specific long-term predictions are not possible for complex systems. This is a result of difficulties associated with even "trac[ing] the impact of any change even after the fact, let alone predict[ing] it ahead of time, making the system complex and hard to control."[180] It is also due to the fact that small changes in complex systems yield big results. Specifically, small changes in the initial conditions of complex

174. *See* PER BAK, HOW NATURE WORKS: THE SCIENCE OF SELF-ORGANIZED CRITICALITY 1-2 (1996).

175. *See id.* at 11.

176. HOLLAND, *supra* note 143, at 12.

177. *See id.*

178. This is referred to as "nonlinearity." "[A]ctions often interact to produce results that cannot be comprehended by linear models. 'Linearity involves two propositions: 1) changes in system output are proportional to changes in input . . . and 2) system outputs corresponding to the sum of two inputs are equal to the sum of the outputs arising form the individual inputs.'" JERVIS, *supra* note 165, at 34.

179. HOLLAND, *supra* note 143, at 31.

180. JERVIS, *supra* note 165, at 17.

systems will be spread throughout the system, multiplied and recycled through flows. These small changes are thus magnified,[181] meaning also that any uncertainty in the initial conditions will be magnified as well.[182] In a complex world, the pretense of long-term prediction must be rejected. The true consequences of our own best actions cannot be known. All that can be done is be locally wise, not globally wise.[183]

D. *Edge of Chaos*

Complex systems are sometimes described as existing "on the edge of chaos," or as a phase transition between order and chaos.[184] This is exemplified in the classes of behavior documented in "cellular automata" by Stephen Wolfram.[185] Cellular automata are simulated collections of cells programmed to carry out rules as a group. "This collection of cells . . . could be viewed as an organism, running on pure logic."[186] Professor Wolfram studied the simplest automata universe—one-dimensional automata arranged in a single line. The initial state was defined at random, and a variety of local rules governing the sites on the automata. "[T]he longtime behavior of the cellular automata" could be

181. WALDROP, *supra* note 142, at 45. It should be noted that not only positive feedback is present in complex systems. Complex systems consist of a mixture of positive and negative feedback lead to complexity. *See id.* at 139. Complex systems, including economies and societies must maintain a balance of order and chaos. "Like a living cell, they have to regulate themselves with a dense web of feedbacks and regulation, at the same time that they leave plenty of room for creativity, change and response to new conditions." *Id.* at 294.

182. *Id.* at 142.

183. KAUFFMAN, *supra* note 143, at 29.

184. *See, e.g.*, WALDROP, *supra* note 142, at 230-31, 292-94; COVENEY, *supra* note 143 at, 271-77; Ruhl, *supra* note 143, at 890-91 (discussing the need of a complex system for a blend of elements of chaos and order).

185. This is also indicated by the "sandpile" experiments of Per Bak and others. He suggests the consideration of a sandpile - first being built on a flat surface by dropping individual grains of sand, leading to a pile. Initially, grains stay pretty much were they land, and disturbances in one area have little effect elsewhere. This is analogous to a linear state. Eventually the slope of the sandpile stabilizes, as newly dropped grains of sand are balanced by sand falling off the edge of the pile. This is like a complex state. Finally, one could imagine dropping sand in such magnitude, or in the appropriate locations so that an "avalanche" of sand would result. This could be considered the chaotic state. *See* BAK, *supra* note 174, at 50-51.

186. COVENEY, *supra* note 143, at 91.

classified "into four types, regardless of specific local rules employed."[187] In Class I the pattern either disappears or becomes static. In Class II "the pattern evolves to a fixed finite size" with continually-repeating patterns.[188] Chaotic states with "little semblance of regularity" occur in Class III.[189] Finally, in Class IV complex patterns occur.[190] These classes of behavior were discovered to bear many similarities to "second-order" phase transitions.[191] Class I and II states are like the ordered states that occur well below the transition point. The Class III state is like the random, chaotic behavior that occurs above the transition point. The Class IV state is like the complex state that occurs near the transition point.[192]

Because ordered and chaotic elements are bound together to create a complex system, it should be noted that alterations in a complex system could serve to drive the system toward either the ordered or chaotic states, just as temperature changes, for example, could lead a system to complete the phase transition, resulting in a fully ordered or fully chaotic state.[193] Thus, it is worth considering what effects these shifts could have on a system. Specifically, the benefits of complexity must be considered, and the detrimental effects of transitions to ordered or chaotic states must be evaluated.

The main consequences of complexity are adaptability and emergence. "[C]omplex systems constructed such that they are poised on the boundary between order and chaos are the ones best

187. *Id*. at 99.

188. *Id.*

189. *Id.*

190. *Id.*

191. *See* WALDROP, *supra* note 142, at 229-30. First-order phase transitions are like the familiar transition of water from ice to liquid at 32° F. The molecule is either in the solid, ordered state, or the chaotic, liquid state. Second-order transitions do not impose a sharp either-or choice on molecules. *See id.* at 229. Rather, at points near the transition (both above and below it), there are substantial regions "where order and chaos intertwine," - in other words, regions of complexity. *Id*. at 230.

192. *See id*. at 230.

193. *See* Ruhl, *supra* note 143, at 891 ("Too many fixed point and limit cycle attractors [characteristics of ordered systems] drag the system into stasis. Too many strange attractors [characteristics of chaotic systems] drag the system into chaos.") *Cf.* WALDROP, *supra* note 142, at 43 (noting that when industries mature, the tend to be more stable, and amenable to description by tradition, linear, economic theory).

able to adapt by mutation and selection. Such poised systems appear to be best able to coordinate complex, flexible behavior and best able to respond to changes in their environment."[194] As described by J.B. Ruhl:

> Systems in the complex region thus exist when the qualities contributing to system sustainability—stability, simplicity, and adaptability—are in harmonious balance, and chaos, emergence, and catastrophe are collapsed into instruments of system evolutionary robustness. Moreover, the blend of attractors needed to promote sustainability necessarily produces emergent behaviors as a result of interaction between the multiple components. Hence, a robust, fit, sustainable dynamical system, because of the inherent presence of some chaos and emergence, necessarily is unpredictable. The key is that the complex systems have turned that source of unpredictability around and channeled it into the trait of adaptiveness, allowing the system to transform disorder into organization.[195]

The value of adaptability and emergence can be most clearly seen by comparison to hierarchical, ordered systems. Such top-down systems of rules of behavior "tend to be touchy and fragile."[196] "[S]ince it's effectively impossible to cover *every* conceivable situation, top-down systems are forever running into combinations of events they don't know how to handle."[197] Although linear systems have the characteristic of predictability, it comes at the price of the emergent properties (the whole being *greater* than the sum of the parts) that exists for complex systems.[198] Thus, ordered systems are less able to deal with future events, and are limited in what they can achieve to merely the sum of the capabilities

194. STUART A. KAUFFMAN, THE ORIGINS OF ORDER 29 (1993).

195. Ruhl, *supra* note 143, at 891.

196. WALDROP, *supra* note 142, at 279. Indeed, studies of traffic jams have found that states with jams of various sizes are better than highly ordered systems with cars going slowly. The latter state would have a higher throughput of cars theoretically, but turns out to be "catastrophically unstable," and "would collapse long before all the cars became organized." BAK, *supra* note 174, at 198.

197. *Id.*

198. *See* COVENEY, *supra* note 143, at 329-31.

of their parts.

Chaotic systems differ from complex systems not so much in behavioral procedures, but in the substantive outcomes of these rules of behavior. Unlike complex systems, "[c]haotic systems have no memory for the past and cannot evolve."[199] Thus, the whole of a complex system may well be *different* from the sum of its parts, but there is no reason to expect that it will be *greater*.[200] With no "memory for the past" the chaotic system cannot create the mechanisms for rule discovery and prediction, or develop evolved tags. Left to itself, a complex system can be expected to maximize its fitness to the extent possible. Chaotic systems can be expected to change, but in ways that do not necessarily bear any relationship to the fitness of the system. In short, the chaotic system cannot maximize its position in the "fitness landscape" - they can *change*, but not necessarily *evolve*.

This is not to say that such perturbations into chaos or linearity are necessarily permanent. Rather, if the system is sufficiently close to the "transition area" the system may evolve back to a complex state.[201] However, the complex state is relatively beneficial, and order and chaos are seen to have relative disadvantages. Thus, it makes little sense to drive the system away from complexity and toward chaos and order, even if the perturbations are insufficient to drive the system fully into ordered or chaotic states.

199. BAK, *supra* note 174, at 30.

200. An example of this "whole is less than the sum of the parts" idea may occur in the case of fibrillation of the heart. When this occurs, the individual muscle cells respond to impulses correctly, yet the heart as a whole "is never all contracted or all relaxed. . . . [T]he parts of a fibrillating heart seem to be working, yet the whole goes fatally awry." JAMES GLEICK, CHAOS: MAKING A NEW SCIENCE 283-84 (1987).

201. *See* WALDROP, *supra* note 142, at 295. To return to Bak's use of the sandpile model, *supra* note 185, he suggests dropping wet sand on the sandpile. This will have greater friction than regular sand. Initially, as the wet sand sticks, there will be smaller, more localized, avalanches. However, this will cause the sandpile to grow steeper, leading the avalanches to grow. Eventually, the pile will return to a state with system-wide avalanches, but with a higher sloped pile. (A similar example could be done with dryer, rather than wetter, sand. The pile would leave the complex "equilibrium" for a while, but eventually return to the critical state, only in this instance with a steeper slope). *See* BAK, *supra* note 174, at 51-52.

III. OPEN SOURCE METHODOLOGY AS A COMPLEX SYSTEM

From the open source projects discussed[202] it is clear that the open source process develops technically strong code. This is true, despite the fact that the development is much more distributed, bottom-up than proprietary methods of development. A consideration of the details of the open source process will show that all the elements of a complex system are present. Thus, complexity theory can explain the observed result of the technical strength of open source projects.

A. *System of Agents*

The open source process consists of many elements that play the role of agents. For example, the underlying technology (hardware and software) fills the role of simple agents. Most obviously, the users, programmers and developers also act as agents in this system. The aggregates of these individuals play important agent roles as well. Project-level communities may act as agents by competing and interacting with other projects. Further, the community as a whole acts as a norm-setting and norm-enforcing agent to help constrain the cultural aspects which are important to the open source process.[203]

The performance systems of the most simple hardware and software agents closely fit the "if-then rule" analogy.[204] Computers, hardware and software literally obey if-then rules and have a performance system dictated by their own programming or materials, acting in concert with their environment (other hardware, computers, programs, users, et cetera). The performance system of the developers is based on their particular computer skills—knowledge of programming languages, programming and debugging abilities, et cetera. The open source community norms pro-

202. *See supra* Section I.A.

203. This is not intended to be an exhaustive list—other agents could be added. For example, individuals who *do not* participate in open source software could be considered agents as well, for the important role that they play in the reward system of the open source community. *See infra*, note 273-274 and accompanying text.

204. *See supra* notes 149-150 and accompanying text.

vide project ownership rules,[205] necessary traits for project leadership,[206] and norms exist that provide a disincentive to engage in self-promotion.[207] The more localized "hurdle" norms[208] are examples of norms for which the relevant community may be a project or group of projects within the larger open source community. It is interesting to note that, although in hindsight analysis these rules strengthen the open source process, there was apparently no centralized effort to develop these rules, as evidenced by the widespread unawareness of their existence.[209]

The credit assignment mechanism for computer hardware and software comes from the changes that result from testing and evaluation done on the code by programmers. The credit assignment mechanism for the norms governing individual and group behavior and rule-choice could be expected to function in a manner consistent with esteem norms generally.[210] However, this category of internal characteristic is particularly difficult to determine based on the information presently available.

The rule discovery mechanism for developers and groups, as with the credit assignment mechanism, should occur in a manner consistent with norms generally. However, the rule discovery mechanism for open source software provides a clearer example of the type of mutation and reproduction typically of complex systems. Specifically, the recycling of code commonly done by open source developers[211] will result in the reuse of successful "rules" in combination with other successful elements of code (reproduction), as well as to serve as the basis for new code development (mutation).

The reuse of code in open source projects is also an example of a type of prediction mechanism. By breaking down a task to be performed by software into familiar parts, pre-existing elements of

205. *See supra* Section I.D.

206. *See id.*

207. *See id.*

208. *See id.*

209. *See* Raymond, *Ownership and Open Source*, *supra* note 41.

210. *See* Richard H. McAdams, *The Origin, Development and Regulation of Norms*, 96 MICH. L. REV. 338 (1997) (discussing the development and nature of esteem norms).

211. *See supra* Section I.B.

code can be used. The role of the normative training received in the academic setting[212] provides one example of the prediction mechanism that occurs in the open source context. The type of behavioral norms learned in the academic setting get carried over and applied to the new activities that occur in open source development. Finally, many of the open source projects, notably Linux, had some basis in existing technology.[213]

B. *Interactions of Agents*

The interaction of agents is one of the most important aspects of complex systems. Thus, the rules for open source agent interaction will next be investigated. Further, the systemic changes that are due to these interactions will be considered. Finally, the means by which open source agents form aggregates is discussed.

1. Rules for Interaction

There are two main rules for agent interaction in complex systems. Both tagging and flows are evident within the open source methodology.

a. Tagging

The first aspect of agent interaction is the use of tags. There are many examples of this behavior in the open source context. One obvious example is the "OSI Certified" certification mark.[214] This mark is able to be used only on software that meets the Open Source Definition.[215] Members wishing to use or participate in the development of "true" open source software can use this "tag" to distinguish open source software from software that only claims to be open, or which may have only a few open source elements.

212. *See supra* Section I.I.1.

213. In the case of Linux, it was initially developed to be like a Unix operating system. *See* Torvalds, *supra* note 12, at 102.

214. *See* Perens, *supra* note 101, at 174 (discussing the creation and use of the open source mark); *The Open Source Page* (last modified Apr. 1, 1999) <http://www.opensource.org/>.

215. *See* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION, *supra* note 3, app. B at 253-54 (the Open Source definition, version 1.0); *The Open Source Definition* (last visited Jan. 30, 2000) <http://www.opensource.org/osd.html> (version 1.7).

Open source licenses can serve a similar function. Although they may be less clear than the simple "OSI Certified" label, the terms of the license can help signal whether a program is truly open source.

Open source community norms also play a substantial tagging role. The norm against self-promotion could be viewed as an interpretation of the "tag" of egotistical behavior as representing someone who will not allocate credit and reputation in the project in a way consistent with community norms.[216] The norms which serve as barriers to entry help identify individuals who wish to participate in code development, but would not be desirable as participants.[217] These norms are tags that allow participants to identify those with whom it is useful to deal and those with whom it is not.

The "gift culture" norms of the community may serve a tagging mechanism in a more subtle way as well. Eric Posner has discussed the ways in which gift giving can have a signaling function that is useful in distinguishing between two types of market actors - opportunists and cooperators.[218] Specifically, giving of gifts, when done appropriately, allows cooperators to distinguish themselves from opportunists in order to pair up with other cooperators.[219] "Cooperators give gifts as a way of showing that they expect a long-term relationship; if they expected only a short-term relationship, they would not obtain a sufficient return to offset the cost of the gifts."[220] This signaling helps to overcome a collective action problem. Although together programmers could engage in activities that produce a cooperative surplus, this can only be achieved through a long-term relationship, while substantial incentives exist to gather short-term benefits and then abandon the relationship.[221]

The use of gift giving as a tag could be true in the open source

216. *See supra* Section II.D.

217. *See id.*

218. *See* Eric A. Posner, *Altruism, Status, And Trust In The Law Of Gifts And Gratuitous Promises*, 1997 WIS. L. REV. 567, 579-80 (1997).

219. *See id.* at 579.

220. *Id*. at 579-80.

221. *See id.* at 581.

community as well. Initiators of a project give a gift of the initial project development and its maintenance, and user give the gift of debugging and development. They signal to one another that they wish to cooperate to take advantage of the benefits that have been seen to accrue to the open source development method. Further, the actions of proprietary software companies claiming to go open source with a program can be observed in this manner as well. If they do not fully conform to the requirements of the open source community (i.e., meet the Open Source definition or other standards) this could help the community recognize a opportunist. Such behavior has been seen in the context of technical standards,[222] and there is potential for its occurrence with open source software as well.[223]

b. Flows

Flows are apparent in open source software development as well. For example, maintainers of an open source project distribute programs and in return receive bug fixes and patches to the program. In return for the work of users, project leaders often get their improvements implemented in future version of the program, and have continued access to the source code to make their own changes. Based on their input, developers receive esteem from other users and group members as well as from the open source community as a whole. Open source businesses receive money in exchange for software and support. This, in turn, may allow for the funding of continued development of open source projects. Proprietary software developers may take programs open source, providing the community with guaranteed access to the source code of the program, in exchange for the benefit of future modifications and the publicity that comes from such a move. Proprie-

222. *See* Marcus Maher, *An Analysis of Internet Standardization*, 3 Va. J. L. Tech. 5 at ¶¶ 18-19 (last visited Jan. 30, 2000)
<http://vjolt.student.virginia.edu/graphics/vol3/home_art5.html> (1998).

223. *Cf.* Leander Kahney & Polly Sprenger, *Apple's Open-Source Movement* (last modified Mar. 16, 1999)
<http://www.wired.com/news/news/technology/story/18503.html> (discussing Apple's claim to an understanding and the beginnings of going open source with their software, and the concerns expressed by Bruce Perens that licenses of IBM and Apple may not fully comport with the Open Source definition).

tary companies may also merely port programs to the Linux platform, gaining a foothold in the open source market, in exchange for providing open source software users with a useful program. Finally, modularity plays an important role in the flows of the system. Modularity of code allows the work of programmers to be spread throughout the system. The reuse of modules from prior programs allow the programmer's work to be spread into future programs. The ease of evaluation and quality improvement that comes from modularity could be expected to make modularity valued by peers.

2. Interactions Yield Systemic Changes

The recycling of code is one good example of how flows result in the spreading of resources throughout the open source system. One piece of code, once introduced into an open source program, can be recycled for use in unrelated programs. These programs may be referenced when still other, different, programs are being written. Recycling might not occur at all for a given piece of code, or it might occur repeatedly into the foreseeable future. The initial input of work by the first author may result in the accomplishment of substantial programming effort when the code's role in future programs is considered.

Part of how flows and tagging create change and diversity in a complex system is through the creation and filling of niches. This process is clearly at work in the open source context. As the range of open source applications increases, niches are created for programs that work with these applications, for adaptations of these programs for particular settings, or for the development of more advanced software that these underlying technologies make possible. External changes, such as those in hardware or standards, also lead to the extinction of former programs and the need for new software to take its place. Thus, the range of products that exist and the change that occurs in the open source community appears consistent with the systemic changes to be expected in complex systems.

3. Formation of Meta-Agents

The fact that meta-agents occur in the open source community

has already been noted.[224] The nature of agent interaction allows for the meta-agents to have capabilities that would be difficult or impossible to achieve in a single sub-agent. For example, the creation of what is now known as the Linux operating system would hardly have been as attainable by Linus Torvalds. Rather, it was the effort of a large number of groups, project leaders and developers that allowed the operating system to reach its current level of sophistication. Similarly, by awarding or withholding esteem based on participation at the aggregate level, critical levels of reputational benefit may be possible, leading developers to participate when such incentive may not have been attainable at the individual or even single-project level.

C. *Specific Example - Linux*

The development of the Linux operating system provides a good example of the complex processes of open source development at work. The agents in this system include Richard Stallman, Linus Torvalds, and others who contributed to the development of the operating system.[225] Agents also include the organizational-level actors, such as the GNU project, which helped orchestrate the development of the specific projects necessary to fill out the operating system's components.[226]

The community guidelines for open source development not only provided a performance system for the participants in the development of Linux, but may in fact have been substantially originated (as discernible principles) in this project's development. These principles came, in part, from the policy statement[227] and legal constraints[228] imposed by the GPL developed by the Free Software Foundation for use with GNU software.[229] This license

224. *See supra* Section III.A.

225. For the roles of various parties, *see, e.g.*, *supra* Section I.A.3.

226. *See* Stallman, *Overview of the GNU Project*, *supra* note 14.

227. The GNU GPL begins with a preamble discussing the justifications for the GPL. *See GNU General Public License*, *supra* note 99.

228. *See supra*, Section I.F.1.

229. *See What is the Copyleft?* (last modified Apr. 11, 1999) <http://www.fsf.org/copyleft/copyleft.html #What is the Copyleft?>.

was utilized by Linus Torvalds when distributing Linux,[230] thus constraining the way in which development could occur. The success of the development of Linux resulted in its development process being mimicked in later open source projects.[231]

Further, the recognized importance of modular code to open source software may also have had its roots in these projects.[232] Indeed, the fact that modularity is widely observed in open source projects is a sign of the success of the credit assignment mechanism. The mechanism credited rules not only based on environmental feedback, but for stage-setting capabilities as well.[233] For example, code recycling was facilitated through modularity.[234] Finally, the choice by Richard Stallman to use Unix as a model for functionality of the operating system,[235] and the decision of Linus Torvalds to base his original kernel on the Minix kernel, which is Unix-like,[236] were instances of prediction mechanisms at work.[237]

Tagging played an important role in the development of the Linux operating system. Confusion over the term "free" lead to a false start in Richard Stallman's attempt to find a free compiler to begin work on an operating system.[238] The "tag" of the GNU GPL

230. Torvalds states that he chose the GPL because that was the license under which the GCC compiler was issued. *See* Torvalds, *supra* note 12, at 107.

231. *See, e.g.,* Raymond, *The Cathedral and the Bazaar*, *supra* note 28 (noting that he chose to use (and write about) the open source development process because of his experience as a contributor to GNU projects, and evidence from the successful development of the Linux kernel that the process could be used on a larger scale).

232. For example, the importance of modularity was recognized by Linus Torvalds, and built into the Linux kernel, *see* Torvalds, *The Linux Edge*, *supra* note 12, at 108. Richard Stallman also recognized the value of modularity, building it into, for example, the Emacs editor. *See EMACS: The Extensible, Customizable Display Editor* (last modified Feb. 16, 1998) <http://www.gnu.org/software/emacs/emacs-paper.html SEC1>.

233. Modular code not only is of greater quality (direct feedback), but allows for easier modifications, project maintenence and recycling of code (stage-setting). *See supra* Section I.E.

234. Increased ease of code recycling is a natural consequence of modularity, due to the greater ease of identifying complete sections of code that perform a particular function, and greater ease in copying only the useful parts of the code.

235. *See* Stallman, *Overview of the GNU Project*, *supra* note 14.

236. *See* Moody, *The Greatest OS That (N)ever Was*, *supra* note 18.

237. These programmers used familiar patterns (the look and feel of the Unix operating system) and used them as a model (either intentionally or implicitly) for what a good operating system should be like.

238. Stallman heard about the "Free University Compiler Kit," or "VUCK," which

under which Linux was released, signaled to GNU that the Linux kernel would be a kernel that would fit with the goal of a free operating system, not only technically, but ideologically. The distribution method of Linux - given away for free public download - may have been the kind of "gift-tag" [239] that would allow GNU to recognize the potential for a long-term relationship with Linux.

Systemic changes and the formation of niches was also evident in Linux's development. A niche was perceived by Richard Stallman - the need for a free operating system.[240] This led to the creation of the GNU Project as well as Emacs and GNU utilities.[241] These formed a complete operating system, except for the kernel.[242] The niche for a kernel was filled by Linux.[243] The existence of the Linux operating system then created further niches for system utilities, user interfaces, file management tools, drivers, et cetera.[244] The filling of these niches create still more niches which must be filled. The existence of these products and a marketplace of consumers led to the filling of the niche by software distributors who added value.[245]

IV. RESULTS OF THE COMPLEXITY OF THE OPEN SOURCE COMMUNITY

The prior section applied complexity theory to the nature of the open source development process and revealed that the open source process is, in fact, a complex. This has several conse-

was designed to handle multiple languages, including C and Pascal. However, Stallman discovered that the university was free, but the compiler was not. *See* Stallman, *supra* note 29 at 57. This shows the imperfection of some tags. The "free" tag was misleading in this case, and has been argued to be misleading for other reasons as well. This lead to a move (by some) the open source tag, which is less likely to cause confusion among as large a group of people. Further advancements may be yet to come.

239. For a discussion of the signaling function of gifts in the open source process, *see supra* Section III.B.1.a.

240. *See* Stallman, *Overview of the GNU Project*, *supra* note 14.

241. *See id.*

242. *See* Stallman, *Linux and the GNU Project*, *supra* note 11.

243. *See id.*

244. *See, e.g.*, Torvalds, *supra* note 12 at 110-11 (discussing future developments needed for future applications of Linux in new settings).

245. *See, e.g.*, Tiemann, *supra* note 129; Young, *supra* note 57 (discussing their business of adding customer service and support to the Linux operating system).

quences for open source software and the development process. First, the complex nature of open source development means that only short-term predictions are possible, particularly with regard to specific matters. Second, and more importantly, the complex nature of open source development makes the high quality software developed by the open source method not only expected, but the natural outcome of the system's complexity.

A. *Short-Term Predictions*

A consequence of the complex nature of open source software development is that the ability to predict the future of open source software or the community with any specificity is impossible. Attempts to predict what software will be developed, what norms will exist, or what the open source marketplace will look like in the mid- or long-term would be foolish. The systemic changes that have been seen in the open source movement when coupled with the fact that small changes in complex systems are magnified through recycling and feedback make such predictions impossible. Further, any change in the surrounding environment facing the open source community will affect the co-evolution of the system in ways not currently predictable.

It is also important to note that the complex nature of a given aspect of any open source project may not continue infinitely into the future. In the study of complexity in economic markets, it is known that as markets mature, they become less complex and more linear.[246] Thus, as some open source projects mature, and fewer changes need to be made, some elements can be expected to stabilize.[247] The niches created by such a product's development can be expected to yield a continued overall complexity, both at the system level, and more specifically at the project level, as new, less mature projects are created to fill these niches.

246. *See* KAUFFMAN, AT HOME IN THE UNIVERSE, *supra* note 143, at 295-96.

247. This is happening with the core elements of Perl. *See* Larry Wall, *Diligence, Patience, and Humility*, *in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION, *supra* note 3, at 127, 140. The same thing is expected with Linux. *See* Torvalds, *supra* note 12, at 110.

B. *Co-Evolution and Fitness Maximization*

The effect of complex co-evolution of software elements such as, user demand and background technology, maximizes the "fitness" of a program, group, community, et cetera, given the constraints of the other elements in the system at that time. By simply relying on the existing open source process, it can be expected that the software developed by the process will be the best[248] attainable given the current position of the open source system in terms of all elements - users, developers, market share, underlying technology, community size, et cetera. That the open source methodology should lead to technically sound products is not unexpected. Rather, it is the natural result of a complex system. It is not just software quality, however, that can be expected to be maximized. Community norms, levels of participation and other factors are regulated by the principles of complexity as well, and can be expected to maximize their relative fitnesses, as they co-evolve along with the software and other elements of the open source environment.

Even beyond the descriptive aspects of complexity theory with regard to open source successes, some general predictions are possible as well. Particularly, it would be the natural consequence of a complex system, such as the open source process, to continue to achieve the maximal attainable fitness in the future. Thus, new versions of existing open source projects, and the new programs developed in the future, can be expected to maintain a high level of technical excellence, as long as the open source methodology remains a complex system of behavior. This should not be mistaken as calling for rigidity. Rather, this conclusion calls for the recognition that the evolution which led to the current open source system can be expected to continue, and that such evolution will result in a system that produces technically superior products.

V. OPEN SOURCE ON THE EDGE OF CHAOS

The conclusion of the last section gives hope for the future of

248. This may not mean best in every category potentially applied to the program. Instead, it means best in an overall sense.

open source software. Such a future can only be expected to occur, however, if the open source approach to software design remains complex. Assuming that the continued production of high quality software is desirable, future action must be undertaken with an awareness of the potential consequences of moving the system from complexity to order or chaos. Short of a complete move of the system out of complexity, admittedly an extreme result, it still makes little sense to engage in activities that tend to push the system toward order or chaos. Although the system will adapt back to a complex balance in many cases, the counter-productive and ultimately fruitless nature of the activities indicate that they should be avoided.

A. *Linearity*

The potential problem for the open source movement from pushes toward linearity comes from several sources. First, pushes towards centralized leadership, even in a limited way, can pose problems for the system. Emphasis on moving open source software into new areas could pose problems as well, if the system has not developed to a stage where it is ready for such a move. Finally, the threat of the closed-source development model could be problematic.

1. The Problem of Centralized Leadership

The paradigm of linear modes of software development occurs in the hierarchical methodologies used by most proprietary software companies. This approach is highly planned, and implemented in a top-down manner. When the open source process moves from complexity toward linearity, a particular threat comes from the natural attraction to centralized explanations experienced by nearly all people.[249] This causes people to impose the concept of a "leader"[250] or a "seed"[251] when explaining phenomena, or

249. *See, e.g.*, MITCHEL RESNICK, TURTLES, TERMITES AND TRAFFIC JAMS 119-44 (1994).

250. In other words, that a phenomenon came about or should come about through the efforts of some centralized command or leader.

251. In other words, that a phenomenon came about or should come about through growth from some preexisting inhomogeneity or element in the environment.

when developing solutions to problems.[252] This tendency may lead people to push the open source community toward linear, centralized approaches to functioning, in ways that could push the system toward linearity.

The most dangerous example of this problem comes from those who would directly impose some hierarchical, top-down leadership on the community. At the moment it is not clear that any organization is openly arguing that it should have total, centralized control over the open source community. However, organizations are arguing for centralized control over certain aspects of the community.[253] The reasoning behind such an approach comes from a failure to appreciate the richness of behavior attainable by complex systems. In the view of such individuals, "there is no place for unintended patterns, arising from decentralized interactions," but rather, an innovative software design or long term success must arise "as an explicit goal."[254] This leads to "misintuitions when people try to make sense of self-organizing systems."[255] In reality, "[i]n many self-organizing systems, random fluctuations act as the seeds from which patterns and structures grow."[256] The apparent randomness in the system actually allows systems to explore alter-

252. *See* RESNICK, *supra* note 249, at 123-24.

253. *See, e.g., Halloween I*, *supra* note 35 (stating that the lack of "visionary leadership" will hold the open source movement back from true innovation); David Bollier, *The Power of Openness - A Critique and a Proposal for The H20 Project* (last modified Mar. 10, 1999) <http://www.opencode.org/h2o> ("the rich latencies of this Internet-facilitated phenomenon may never develop if a new kind of networking leadership does not coalesce to assert the important values that can only flourish in an environment of openness" which, he proposes, could be filled by a new organization "H2O"). Another variation on this theme are those who provide a list of ways to make money on open source projects. *See, e.g., The Business Case for Open Source*, *supra* note 129; <http://www.opensource.org/for-suits.html>; Hecker, *Setting Up Shop*, *supra* note 75; *Halloween I*, *supra* note 35. While for the most part, these are merely lists of approaches that have been successful thus far, there is a danger that, implicit in reading the list could come the idea that these are the only categories of activity, and one can follow them, or forget about making money. In reality, any effort to provide some wooden, formalistic business models, no matter how clever, will be unable to adapt and evolve to meet future challenges.

254. RESNICK, *supra* note 249, at 125.

255. *Id.* at 137.

256. *Id.*

natives in parallel and reach a "'global' optimum."[257] Because the system behaves differently at different levels of abstraction,[258] such an optimum may not be apparent, let alone attainable from a top-down perspective. Thus, an approach that appears good from a linear perspective may be too fragile to survive, or may miss out on opportunities that arise from nonlinear effects that are, almost by definition, not predictable by a "leader." Although it may appear riskier to those unfamiliar with complex systems, leaving innovations of all kinds to the system itself may be the best approach to preserving the complexity that has made open source software good.

A natural point of criticism of this argument is the observed importance of project leaders in the open source process.[259] Indeed, the argument that linear traits are being discovered in, or proposed for, a complex system need not immediately signal a problem—complex systems involve balances of order and chaos.[260] However, the potential "harm" resulting from a given project leader is limited to the replicability of the project. If the leader is taking the project in a direction that some nontrivial segment of users dislike, they may fork the project and create their own version to compete with the existing project. They could even go so far as to create an entirely new project. This could be done relatively easily (assuming user interest) for smaller projects. For larger projects (for example, Linux) it might be reasonable to expect greater difficulty in "forking" the overall project, due to the effort required to reproduce an operating system. However, big projects are often subdivided into smaller projects, which are more susceptible to forking. It is further worth noting that the difficulty of reproducing a program of the scope of a large project is far from prohibitive.[261] Thus, the "linear" nature of project leadership is

257. *Id.* at 138-39.

258. In other words, meta-agents have different properties than their sub-agents or the mere sum of these agents, and meta-meta-agents have different properties than meta-agents, et cetera. *See, e.g., id.* at 139-41.

259. For a discussion of the role of project leaders, *see*, *supra* Section I.C.4.

260. *See supra* Section II.D.

261. *See, e.g., GNOME Project* (visited Apr. 8, 1999) <http://www.gnome.org> (project to develop a window manager).

constrained, in part, by the "chaos" of code forking.[262]

The norms of the open source community have also evolved to constrain the role of project leader.[263] These norms regulate the project leader to require behavior in a manner consistent with the complexity of the open source system. The means of assigning credit to contributors, norms against self-promotion and the obligations required before taking over a project all facilitate the overall process which allows for distributed, complex behavior.

A further counter-argument that could be raised against the critique of top-down leadership is that businesses or the government expect, and feel most comfortable dealing with, such an entity.[264] That businesses and governmental authorities should have such expectations is not surprising.[265] However, to simply cave in to those desires evidences a failure to understand and apply another basic principle of complex systems - that the environment co-evolves along with the complex system.[266] This gives rise to several points. First, due to *co*-evolution, when the open source community creates a centralized model, the preference in *business and government* for a centralized model could have the effect of making the *open source community* more centralized.[267] The corollary to this point is that, by using centralized bodies to deal with government and business groups, the opportunity to force these bodies to change in order to understand open source development may be lost.[268] Finally, even if the goal of the open source organization is

262. For a discussion of code forking as a chaotic system, see *infra* Section V.B.

263. *See supra* Section I.D.

264. Indeed, Eric S. Raymond makes almost exactly this argument in support of his proposed position of "open source evangelist." *See* Eric S. Raymond, *The Revenge of the Hackers*, *in* OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION, *supra* note 3, at 207, 213-15.

265. As noted, *supra* notes 249-252 and accompanying text, the tendency to look for centralized causes, processes and authority is a learned behavior common to nearly all people. Indeed, even individuals with substantial experience researching and studying complexity harbor such tendencies. *See, e.g.*, RESNICK, *supra* note 249 at 119-20 (discussing such behavior in Marvin Minsky, who "has though more—and more deeply—about self-organization and decentralized systems than almost anyone else.")

266. *See, e.g.*, WALDROP, *supra* note 142, at 259-60; HOLLAND, *supra* note 143, at 97; RESNICK, *supra* note 249, at 142-44.

267. For example, through the creation of the new organizations.

268. Resnick notes that "[b]eing taught a list of rules isn't going to have much ef-

not to be a community leader, but merely to inform business and government about the merits of open source software, the tendency of business and governmental bodies to seek centralized explanations will likely lead them to treat the open source organization as a leader. This will result in an ultimate failure to understand the true nature of the open source process.[269] Thus, it is important to weigh carefully not only the effect a leadership organization would have on the open source community, but the effect it would have on its environment as well. The environmental effects of today will lead to the co-evolutionary changes in the open source movement of tomorrow.

2. Problem of Pushing Open Source into New Areas

A failure to understand the co-evolutionary relationship of the open source community and its environment can lead to other potential threats to the complex nature of the system. For example, taking for granted the high technical standards that open source software has thus far been able to attain could lead to mistaken initiatives. It could lead to initiatives to push open source software into new forums for which it may not be useful, and in fact, for which the complex approach may not necessarily be appropriate. Specifically, attempts to make particular open source programs the standard for everyone[270] - techies and non-techies alike - may be misguided.[271] Complex systems are built upon the system's his-

fect on a firmly entrenched centralized mindset." RESNICK, *supra* note 249, at 147-148. Rather, moving beyond the centralized mindset comes from "participat[ing] in a culture that values and encourages decentralized thinking." *Id*. at 148.

269. Resnick refers to a similar experience in the area of the sciences. "Just as children assimilate new information by fitting it into their preexisting models and conceptions of the world, so do scientists. . . 'In short we risk imposing on nature the very stories we like to hear.'" *Id*. at 122.

270. *See generally*, Raymond, *The Revenge of the Hackers*, *supra* note 264. Underlying his statements that the open source community had "failed," or was "losing," and his goals for the future is the idea that "winning" means maximized adoption of open source software.

271. Just as another example, the idea that "freedom" arguments should be abandoned due to their claimed inability to recruit businesses to open source software, *see, e.g., id*. at 212, mistakenly assumes that the open source culture, which is so important to the success of the process, would unaffected by the abandonment of the very principles that led many participants to join the community in the first place.

tory - what is possible in the system now is a result of where the system was at prior points in time.[272] This means that what will be possible in the future is dependent upon intervening events. To move the open source system along more rapidly toward a goal it may not achieve for some time (or may never achieve) is to reject the complexity that lead to the best qualities of the open source software.

The potential for mistaken initiatives applies to the issue of linearity in an important way. To understand how, it is first important to consider the issue of free riders with regard to open source software. Commonly referred to as a free-rider "problem," the effects of free riders on open source development can vary from good to bad. Some level of free riders[273] is good. The esteem or respect awarded a developer is dependent upon the baseline of participation.[274] If everyone is participating at a high level, for the developers to get respect they must participate at an even higher level. However, they may alternatively choose not to participate at all, if the esteem is not worth the effort that would be required. If there are free riders, then the baseline for respect is much lower, and participation at even low levels will garner peer esteem. Thus, it is reasonable to expect the system to maintain a rough balance between the number of participants and free riders. Too many free riders will lower the effort required to get esteem, leading to more participation. Too much participation will raise the bar so high that some participants will drop out, increasing the number of free riders.

The problem in this situation comes from the fact that, while the number of potential free riders is unlimited, there is an upper limit on the number of participants. In particular, the non-hackers that are intended to be reached by the widespread adoption of, for example, Linux, may reasonably be expected to have little or no capability to participate as developers. Further, these users have a much lower ability to appreciate good programming, and thus less

272. *See* JERVIS, *supra* note 165, at 40.

273. In the open source context free riders will refer to users of open source software who do not contribute, or those who neither use open source software nor contribute.

274. *See* McAdams, *supra* note 213 at 365-67.

ability to award esteem.[275] This particular element of the market may be unable to be anything but free riders on the basic open source model. What these users do have to offer in terms of compensation is money. Thus, by purchasing software they could fund an open source project that would create software that met their specific needs. However, this process (software for money) is much more linear in nature than the complex system of project leader-user/developer interaction and feedback that leads to the complexity of the current open source method. Non-techie users would be less attentive to the tagging mechanisms. Since they would not be making changes to the source code, they would be less likely to care about the Open Source certification mark, open source licenses or norms that help distinguish cooperators from opportunists. It would be reasonable to expect no flow of recycled code, and they would likely be much less active in the niche-filling that occurs as the result of new product development.

Having a linear model of software development for these users is only a potential outcome. There is no empirical evidence of how little these users would contribute, or how much contribution is actually need. However, two important points can be derived from the forgoing analysis. First, linear models of software development may be entirely appropriate for some market segments.[276]

275. Since they have no technical knowledge it is difficult for them to evaluate the product, because they have only a limited baseline of comparison. Another way of saying this, is that their esteem should be less valuable (although probably greater than zero) because they do not have the knowledge necessary to allocate it in the manner consistent with their own best interest (maximum reward for the programming that is best, and thus helps them, minimal or now reward for bad programming which does not help them much). *Cf. id.* at 361-62 (discussing the importance of the ability to detect norm non-compliance to the ability to effectively grant or withhold esteem).

276. This is where the arguments of Richard Stallman, the Free Software Foundation and others regarding the freedom that comes from open source software become particularly important. Although the linear method of software development is commonly associated, even within this paper, with proprietary software development, it need not be. Linear methods could also follow the same licenses as open source products, but with little anticipated availment of the source code access by users. However, the economic focused arguments do not indicate why open source should be preferred if a linear model can develop as high quality of software and meet the needs of this market segment. However, if there are freedom considerations associated with free software, it is worth pursuing, even in the linear context, where the users have not practical (i.e., improvement in software quality) benefits from doing so.

Second, the tendency of the segment toward linear development models, to the extent it occurs, could needlessly interfere with the complexity of the regular open source model. It could be preferable to develop this technology as a separate entity, if possible, to avoid any "pollution" of the existing, complex, open source approach.

3. The Problem of Closed-Source Development

Although, as noted in the prior section, a linear approach to software development need not be closed-source, the counter-proposition may be true. That is, it may be the case that proprietary software development inherently follows a centralized, linear mode. Much of the ability of proprietary developers to effectively control use of their copyrighted material comes from limited access to this material.[277] Thus, the closed-source development model almost certainly requires a barrier to the user/developer feedback and interaction typical of the open source model. This will also mean that the recycling of code is likely to be limited to "in-house" developers, if such recycling occurs at all. Similarly, the only flows occurring in the proprietary software system will likely consist of the money-for-software exchanges. Finally, interactions are much more rarely likely to cause changes in the system. While customers can kill a product they have no desire for, they have almost no ability to exert influence on the characteristics of the products available to them.[278] The centralized control neces-

It is also interesting to note that Netscape's Navigator may be similar, from the perspective of complexity vs. linearity, as operating systems. The Mozilla open source project has had only about 30 voluntary participants as compared to about 100 paid programmers. The project is also months behind in its first beta release. *See* Ben Elgin, *Open-Source Mozilla Project Struggles for External Support* (last modified Apr. 5, 1999) <http://www.zdnet.com/pcweek/stories/news/0,4153,1014274,00.html>. These factors indicate that the Mozilla project may not be reaping the expected open source benefits. However, the ultimate conclusions that can be drawn at this point are merely speculative.

277. *Cf.* Mark Stefik & Alex Silverman, *The Bit And The Pendulum: Balancing The Interests Of Stakeholders In Digital Publishing*, 16 No. 1 COMPUTER LAWYER 1 (1999) (discussing the importance of physical constraints on copying for effective control of copyrighted material, and the potential role to be played by trusted systems in the digital world).

278. *See The Customer Case for Open Source* (last modified Nov. 28, 1998) <http://www.opensource.org/for-buyers.htm>.

sary to bring about the secrecy critical to protection of intellectual property points to a linear development model as the natural consequence of proprietary software development.

4. Intellectual Property Problems

The link between closed-source software development and linearity can pose problems for open source software, in particular, through intellectual property. By invoking copyright or patent restrictions in code,[279] proprietary companies can attempt to impose their centralized model of development on the open source community by forcing them to go through the proprietary company for access, or to find some route around the protected code. Further, the open source movement must be careful to ensure that projects remain open source through future iterations, and are not cut off from the open source community at some future date through the use of intellectual property. This calls for the use of open source licenses, such as the GPL, that do not allow future versions or modifications to be "taken private."

To ensure that open source projects are not taken private also requires that the license originally attached to the program "apply to all to whom the program is redistributed without the need for execution of an additional license by those parties."[280] The validity of this type of open source license has not yet been confirmed by any court.[281] Its validity may be supported by recent cases upholding the validity of roughly similar "shrinkwrap" licenses.[282]

279. S*ee,* Bruce Perens *Preparing for the Intellectual Property Offensive* (visited May 11, 2000) <http://www.linuxworld.com/linuxworld/lw-1998-11/lw-11-thesource.html>.

280. Perens, *The Open Source Definition*, *supra* note 101, at 179.

281. *See id.*

282. *See, e.g.*, Hill v. Gateway 2000, 105 F.3d 1147 (7th Cir. 1997), *cert. denied* 118 S.Ct. 47 (1997) (upholding arbitration clause in contract for hardware and software, presented to the user upon receipt of computer)*;* ProCD v. Zeidenberg, 86 F.3d 1447 (7th Cir. 1996) (enforcement of a shrinkwrap license included with software); Brower v. Gateway 2000, 676 N.Y.S.2d 569 (App. Div. 1998) (upholding arbitration clause in contract for hardware and software, presented to the user upon receipt of computer); *but see* Step-Saver Systems, Inc. v. Wyse Technology, 939 F.2d 91 (3d Cir. 1991) (shrinkwrap license unenforceable as not part of contract terms made in an earlier phone conversation). *See also*, Ira V. Heffan, Note, *Copyleft: Licensing Collaborative Works in the Digital Age*, 49 STAN. L. REV. 1487, 1509-11 (1997) (arguing that open source licenses

The outcome of a new proposed uniform state act, the Uniform Computer Information Transaction Act (UCITA),[283] may play a substantial role in open source license validity as well. UCITA supports the validity of mass market licenses generally.[284] As currently written, if no specific duration of a license is specified, the default is a perpetual license.[285] This is consistent with the understanding in open source licenses. However, source code licenses within the proprietary software community are rarely for a perpetual term. Thus, the UCITA, if adopted by the states, could help resolve some of the uncertainty regarding open source licenses.

B. *Chaos*

Although there are a wealth of problems that could push the open source model toward linearity, there are also problems that could push it toward chaos. Specifically, these problems include code forking, the failure of project leaders and incompatibility of open source licenses.

1. Code Forking

Strong norms against formal code forking[286] exist in the open source community.[287] However, the marketing of open source software to businesses stresses the ability of companies to change the source code themselves. Many of the code changes may be passed on to the seller based on the hope of incorporation into future releases of the product. There is also the potential for busi-

should be found valid under existing law).

283. *See Uniform Computer Information Transaction Act* (last modified Aug. 4, 1999) <http://www.law.upenn.edu/bll/ulc/fnact99/1990s/ucita.htm> [hereinafter UCITA].

284. *See id.* § 210.

285. *See id.* § 308(2)(B) (". . . the duration of the license is perpetual as to the contractual rights and contractual use restrictions if: . . . the license expressly granted the right to incorporate or use the licensed information or informational rights with information or informational rights from other sources in a combined work for public distribution or public performance."). The GNU GPL, for example, expressly allows modifications of GPL'd programs to be distributed themselves, or as part of a new program, so long as the new work is released under the GPL, thus falling within the terms of this provision.

286. What is meant by "formal" code forking is the creation of a new, competing open source project based on the same underlying source code.

287. *See, supra* Section I.D.

nesses to simply keep these changes for themselves. It is not necessarily the case that they would do so out of a desire to market the product themselves, but rather, simply because there was no institutional awareness that these changes should go back to the community, or that there would be any value to resubmitting them.

If such a tendency were present, over time the products being used by businesses would develop vast numbers of informal "versions" of the product. Essentially, the project would no longer have a "memory of the past" or the same ability to evolve that is expected of complex systems. Rather, as is common in chaotic systems, the code would develop along one path, with little to no understanding of the history of development within each of these smaller versions. These varying versions may propagate as future employees alter their own version of the software to deal with hardware and software changes, or as the code is passed between companies through merger or buyout. Typical of chaotic systems, these programs will change, but not necessarily in an "evolutionary" manner. Rather, the changes will bear only a limited relationship to the fitness of the code given the current environment.

To avoid this result, it is important to continue supporting the norms against code forking. However, more is obviously needed. Sellers of open source products must also be involved in the code's development. This will allow the feedback of customers to be incorporated into the formal project, giving businesses greater incentive to submit code changes, and allowing the project to "learn" from these changes. Maintaining the quick release rate that is currently common to open source projects is important as well. The rapid release rate will help ensure that projects do not have enough time to evolve too far along different paths in individual companies. It will also encourage businesses to submit changes to avoid the need to constantly fix and re-fix new version of the program to deal with their particular environments.

2. Poor Project Leadership

It is important to reiterate the role of project leaders in the success of open source projects, particularly in light of the criticisms of other types of leadership in the prior section. Indeed, just as the chaotic nature of code forking helps balance the potentially linear

nature of project leadership, project leadership is necessary to balance chaotic potentials of the open source model. First, there is the obvious potential for increased code forking due to the absence or inaction of project leaders. It was noted[288] that a disbelief in the potential for users' changes to be implemented in new versions of the software could lead the users to keep the changes to themselves.[289] This was shown to potentially lead to numerous unique version of the software, resulting in a more chaotic open source project.

Excessively weak project leadership can lead to failure on the part of users to resubmit contributions and an increase in the chaotic nature of the system. Specifically, this could come from a lengthening of the intervening time between releases. Part of the strength of open source software comes from the rapid updating of project versions.[290] This was seen to encourage contribution of users' individual changes back to the project. Failure to make final decisions regarding disputes over the proper course for the project, which is part of the leader's role, could slow the release of new project versions as there is an attempt to achieve some consensus regarding the technical development.[291] This may be part of the reason why the "benevolent dictator" model of project leadership is seen to be more stable and less complicated than the committee model.[292]

3. Inconsistent Open Source Licenses

A final problem that could lead to greater chaos in the open source community is inconsistent open source licenses. "The propagation of many different and incompatible licenses works to the detriment of Open Source software because fragments of one

288. *See supra* Section V.B.1.

289. For example, when one of the GNU tools, GDB, went through a period with "no strong maintainer," this resulted in GDB fragmenting "with hundreds of people around the world making their own versions to meet their own needs." Tiemann, *supra* note 129, at 79. When this was finally taken over by a Cygnus engineer he collected 137 versions of the program that required integration. *See id.* at 81.

290. *See supra* Section V.B.1.

291. *See, supra* Section I.C.4.

292. *See id.*

program cannot be used in another program with an incompatible license."[293] The inability to use a given program in conjunction with others or as part of a package could result in several types of problems. First, if the program fills a niche necessary to the package, it will be necessary to develop a new, equivalent, product. Essentially, a type of forking must occur to create a product that can be released under a compatible license. Further, if sufficient software exists under incompatible licenses, a number of packages might arise, based upon the number of incompatible open source licenses. Thus, the forking will be repeated for each of the packages that are available.

Inconsistencies in open source licenses can further lead to an inability to take advantage of lessons learned in past open source projects. Specifically, the practice of recycling code from prior open source projects could be substantially hindered by inconsistent license terms coupled with uncertainty regarding a potential finding of copyright infringement by a court.[294] This could discourage the recycling of code from other programs if the license under which it was issued is inconsistent with the license under which the current program is to be distributed.[295]

293. *See* Perens, *The Open Source Definition*, *supra* note 101, at 185.

294. For example, the consequence of translating a program for a new platform is not entirely certain. *Compare* Whelan Assocs. v. Jaslow Dental Laboratories, 797 F.2d 1222 (3d Cir. 1986) (holding that translating a program from one source code to another to allow the program to run on a different platform was a direct copying copyright infringement) *with* Q-Co Indus., Inc. v. Hoffman, 625 F.Supp. 608 (S.D.N.Y. 1985) (where a program was written in BASIC, a program substantially similar in function and screen design written in PASCAL was not an infringement). Direct or literal copying can constitute infringement even if only a small portion is copied. *See, e.g.*, Susan A. Dunn, Note, *Defining The Scope Of Copyright Protection For Computer Software*, 38 STAN. L. REV. 497, 512 (1986) (noting that, in one case, SAS Inst., Inc. V. S&H Computer Sys. Inc., 605 F. Supp. 816, 822, 830 (M.D. Tenn. 1985), infringement was found due to literal copying of less than one fortieth of one percent of a program's code). Thus, the translation of an open source program for a new platform and subsequent release under an incompatible license could be deemed an infringement, as could the copying of some small portion of code then used in a program released under an incompatible license.

295. For example, if code from a GPL-licensed program were used in a program released under an incompatible license, such use would fall outside the license and would subject the developer of the later program to potential liability for copyright infringement. This is true because the user of a GPL program has a license to use and re-release parts of its code "under the terms of [the GPL] License." *See The General Public License*, *supra* note 99. While it might also, generally, be necessary to consider whether

Ultimately, this poses less of a potential problem than the others in this section. A good open source definition can help ensure minimal conflict among compliant licenses. It is also unlikely that there would be a sufficient number of programs within each of the license categories to lead to competing packages. Nonetheless, there exists the potential for such a situation to develop, leading to greater chaos in the open source process.

VI. IMPACT OF COMPLEXITY ON FUTURE CONCERNS

The prior sections have shown both the benefits that have already been found as a result of the complexity of open source software development, and the ways in which this complex nature may be maintained. So long as the practices necessary for the success of the open source process are continued, there is every reason to believe that many of the concerns expressed about the future of open source software are unwarranted.

A. *Reaction of Microsoft*

One of the biggest issues facing the open source community is the potential reaction of Microsoft to counteract the success of open source software.[296] Microsoft can be expected to use all the

the license allowed for the creation of derivative works by anyone other than the owner, the Open Source definition effectively nullifies this for all open source software by requiring that Open Source licenses allow for user modifications. *See The Open Source Definition* (last modified Dec. 20, 1998) <http://www.opensource.org/osd.html> ("The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.")

296. Microsoft has itself expressed the view that Linux poses a threat. *See Halloween I*, *supra* note 35 (internal Microsoft memo discussing the nature and threat of open source software generally); *Linux OS Competitive Analysis* (last modified Aug. 8, 1998) <http://www.opensource.org/halloween/halloween2.html> (hereinafter "*Halloween II*") (followup to *Halloween I*, discussing threat of Linux to Microsoft); Chris Oakes, *MS: Open Source is Direct Threat* (last modified Nov. 2, 1998) <http://www.wired.com/news/technology/0,1201,15990.html> (noting Microsoft's acknowledgment of the validity of the Halloween memos, and recognizing the threat to Microsoft posed by open source software); Nicholas Petreley, *Take Nothing for Granted* (visited May 19, 2000) <http://www.linuxworld.com/linuxworld/lw-1998-12/lw-12-penguin.html> (stating expectation that, after antitrust lawsuit ends, Microsoft will focus its efforts to stopping open source software to a much greater extent). *But see* Scott Berinato, *Microsoft Exec Dissects Linux's 'Weak Value Proposition'* (last modified Mar. 4, 1999) <http://www.zdnet.com/pcweek/stories/news/0,4153,1014079,00.html> (noting the

tools at its disposal to fight the open source movement - including code, norms, the market and law.[297] In the Halloween memos it expressed an intent to use code by taking highly commoditized, simple protocols and "extending these protocols and developing new protocols, [to] deny OSS projects entry into the market."[298] Further, although the memos indicate a concern that Microsoft's FUD[299] tactics may be ineffective on open source software,[300] its use of tactics of some kind[301] to get user and developer mindshare can be expected. Finally, Microsoft may well attempt to utilize its market position to force software vendors and OEMs to choose between Microsoft and Linux.[302] The Halloween memos indicate that law may be used by Microsoft as well, in particular, the use of

statements of one Microsoft executive that further studies of Linux indicate it is less of a threat).

Although Linux, rather than open source software generally, would appear to pose the greatest threat to Microsoft, their own memos indicate an intent to "target a process rather than a company." *See Halloween I*, *supra* note 35.

297. For a discussion of the regulatory power of these forces *see, e.g.*, *See, e.g.*, Lawrence Lessig, *The Constitution of Code: Limitations on Choice-Based Critiques of Cyberspace Regulation*, 5 COMMLAW CONSPECTUS 181 (1997); Lawrence Lessig, *Constitution and Code*, 27 CUMB. L. REV. 1 (1997) (discussing the regulatory nature of law, norms, market and code).

298. *Halloween I*, *supra* note 35.

299. FUD stands for "fear, uncertainty and doubt."

300. *See Halloween I*, *supra* note 35.

301. *Halloween I* provides an indication that Microsoft would attempt to offer some of the "community"-type benefits of open source software, including "putting out parts of the code base" or "creating community/noosphere," although not fully embracing the open source concept. *Halloween I*, *supra* note 35. Essentially, this would involve the creation of Microsoft's own, proprietary "open source" methodology, thus taking a page from their technique of creating Microsoft's own "standards" in place of true technology standards. Some indication of this has come from statements by Microsoft that they may be considering taking their code "open source," but then noting that they had a unique definition of "open source." *See* Connie Guglielmo, *Microsoft to Open Source? Not Likely* (last modified Apr. 9, 1999)
<http://www.zdnet.com/zdnn/stories/news/0,4586,2239301,00.html>.

302. *See, e.g.*, Petreley, *Take Nothing For Granted*, *supra* note 296 (predicting that, after the DOJ antitrust suit, Microsoft will force vendors to choose between Linux and Microsoft NT just as they did for Navigator and Internet Explorer). There is historical precedent for such activity. *See* United States v. Microsoft Corp., Complaint, Civil Action No. 98-1232 (visited Feb. 26, 1999)
<http://www.usdoj.gov/atr/cases/f1700/1763.htm> at ¶¶ 24, 97 (use of market power to control OEM alteration of desktop); *id.* at ¶¶ 75-92 (use of market power to control Internet content providers).

patents and copyright for "combatting Linux."[303]

The tactic of Microsoft to de-commoditize technical standards can be successful only if it achieves sufficiently widespread implementation of its modified technology to make its version the de facto standard. However, for many of the technologies for which this would be necessary, Microsoft may not have sufficient market share to do so. With network technologies, the participants in the IETF, World Wide Web Consortium (W3C), and other Internet standards organizations could oppose Microsoft's competing, proprietary "standards." Other companies will have incentives to fight Microsoft on this front as well.[304] Thus, at least within the server market, where Microsoft has less than 50% market share,[305] it is unlikely that sufficient market presence exists for it to force a de facto standard.

Such an attempt may also have the effect of garnering greater support for the open source community, given the dim light in which such anti-competitive conduct is viewed, particularly because of Microsoft's recent antitrust problems. Although not a necessary trait of the open source community, there seems to be an

303. *Halloween II*, *supra* note 296.

304. For example, Sun filed a lawsuit against Microsoft for allegedly engaging in this type of "decommoditizing protocols" with respect to Java. *See Microsoft Readies Java Appeal* (last modified Dec. 17, 1998) <http://www.wired.com/news/news/politics/story/16895.html>; Will Rodger, *Sun, MS Play Java Blame Game* (last modified Dec. 10, 1998) <http://www.zdnet.com/zdnn/stories/news/0,4586,2174437,00.html>; Will Rodger, *Java Emerges as Key Antitrust Issue* (last modified Dec. 4, 1998) <http://www.zdnet.com/zdnn/stories/news/0,4586,2172096,00.html>. It has further been noted that in the Internet market of the future, there will be a greater diversity of competitors, and the market may not readily facilitate a single monopoly player. *See* Eric Nee, *Microsoft Gets Ready To Play a New Game*, FORTUNE, Apr. 26, 1999 at 106. Additionally, much of the software that provides the functionality of the Internet is itself open source (for example, Sendmail, BIND and Apache), and these projects would have incentives to resist changes to implement proprietary "standards." Indeed, to the extent that the open source licenses under which these programs are released restrict the licenses of software that it uses, or with which it can be aggregated, use of these proprietary standards could be impermissible.

305. *See* Stephen Shankland, *Linux Shipments up 212 percent* (last modified Dec. 16, 1998) <http://www.news.com/News/Item/0,4,30027,00.html> (citing statistics from an International Data Corporation study indicating that market share for Windows NT was about 36%, while Novell Netware, for example, had about 24%, Linux had about 17%, and Unix accounted for roughly another 17%).

impression that anti- Microsoft sentiment plays a role in the interest and participation of some in the open source movement.[306] Further anti-competitive conduct would reinforce the perception that open source software, and Linux in particular, is a legitimate alternative to Microsoft.[307] This would also support the conclusion that Microsoft readily engages in questionable, anti-competitive conduct. Thus, efforts by Microsoft may well be unable to provide sufficient code constraints to inhibit open source software, while having the effect of reinforcing pro-open source opinions.

Efforts to constrain OEMs and software developers through the use of Microsoft's market position, to the extent it would have an effect, could also be frustrated. If Microsoft is successful in controlling OEMs and software vendors, this could drive innovation in the open source community. With no commercial options available, Linux applications and utilities could be expected to be developed by the open source movement to fill the gap caused by Microsoft's control.[308] Further, if Linux is not able to be preinstalled on machines, this would provide strong incentives for innovation among commercial Linux distributors to provide better, more user-friendly instillation tools. Alternatively, feedback effects from the current trend of commercial software application developers porting software to Linux[309] could lead to increasingly steep growth in the market. This could lead to a larger market for open source software, providing revenue to fund more develop-

306. *See, e.g.*, *id*. (citing anti-Microsoft sentiment as part of the reason for Linux's growth); Bob Sullivan, *Linus Torvalds—Microsoft Killer?* (last modified Feb. 8, 1999) <http://www.msnbc.com/news/239469.asp> (characterizing Linus Torvalds as the "patron saint of anti-Microsoft forces"); *Halloween I*, *supra* note 35 (stating that the ability to capitalize on anti-Microsoft sentiment is a strength of Mozilla).

307. This perception would be reinforced by the fact that Microsoft thought it was a serious enough threat to bother to attack it with anticompetitive behavior.

308. *Cf.* Behlendorf, *supra* note 44, at 160 (noting that there is pressure to bridge gaps in open source software with more open source programs).

309. *See, e.g.*, Charles Babcock, *Database Vendors, Netscape Support Linux* (last modified July 27, 1998)
<http://www.zdnet.com/zdnn/stories/news/0,4586,339810,00.html> (Informix, Oracle and Netscape make software on Linux); Ben Elgin, *Corel Chooses Linux OS for Its NC Prodcuts* (last modified May 8, 1998)
<http://www.zdnet.com/zdnn/content/smro/0508/314997.html> (Corel to make network computer products available on Linux).

ment, and further reinforcing the status of the community.

The challenges posed by the use of intellectual property as a weapon clearly could have serious consequences for the open source community.[310] One outcome of such a challenge would be forcing to the forefront a consideration of the incentives and motivations for the existence of intellectual property in the online world. A potential resolution of these issues could lead to Microsoft's ability to restrict the use of code by open source programmers. This would test, once and for all, the proposition that the open source movement is unable to engage in innovation.[311] Complexity theory suggests that the community would respond by the development of innovative software technologies that would allow the circumvention of Microsoft's patents and copyrights. Such innovation may convince outsiders of the full extent of open source's potential, more than compensating for the loss due to legal restrictions on code.

An attempt by Microsoft to create its own version of the "open source" concept could be dealt with by the open source community as well. Attempts to co-opt the open source term provides both the opportunity to reinforce community standards regarding open source requirements, as well as educate others through the forum provided by the media coverage. Indeed, an attempt to dilute the open source requirements by Microsoft might be more likely to elicit a strong, immediate response[312] from the open source community than similar conduct from a less-vilified software developer. Attempts to formally attach the "Open Source" term to its software could also lead to a trademark suit against Microsoft,[313] further reinforcing the true requirements of open source software

310. This is the area of attack with the least certain outcome for the open source community. For a discussion of the concerns associated with intellectual property restrictions, *see, supra* Section V.A.4.

311. This claim is raised, for example, in *Halloween I*, *supra* note 35.

312. For example, on Slashdot <http://www.slashdot.org>, an open source news service, Microsoft's statements that it had its own definition of the term "open source" led to the posting of nearly 300 comments within about one day. *See Microsoft Redefines Open Source* (visited Apr. 15, 1999)
<http://slashdot.org/articles/99/04/09/1850204.shtml>.

313. This is due to the fact that the term "Open Source" has been made a certification mark. *See supra* note 97.

and providing yet another forum for introducing this concept to others. Thus, the response of the open source community to these attacks could strengthen the movement as a whole.

B. *Clash of Norms*

Open source cultural norms are critical to the success of open source software.[314] However, efforts to "evangelize" the open source cause has led to a clash of norms between what is accepted in an online environment and what is the traditional behavior in the "real world." This conflict takes two forms. The first type involves individuals engaged in behavior that is consistent with the norms of the offline world, but seem violative of certain online norms.[315] The alternative problem occurs when behavior that is consistent with online norms is misinterpreted according to the norms of the offline world.[316] These conflicts could lead to several problems. Notably, open source community norms could be contaminated by these alternative norms,[317] or the impression of the

314. *See supra* Section I.D., Section III.

315. Eric S. Raymond recognized that the publicizing of the open source movement in traditional media could lead to the appearance that he was violating the norms of the open source culture. *See* Raymond, *The Revenge of the Hackers*, *supra* note 264 at 214 ("I'd probably end up . . . despised as a sell-out or glory-hog by a significant fraction of [the open source community].")

316. One situation that may well be an example of this involves the dispute between Eric S. Raymond and Bruce Perens regarding the consistency of Apple's open source claims and the requirements of the open source definition, and subsequent war of words. *See* Leander Kahney, *Open-Source Gurus Trade Jabs* (last modified Apr. 10, 1999) <http://www.wired.com/news/news/technology/story/19049.html>. While this heated dispute appears like real dissension among these individuals, Chris DiBona, director of Linux Marketing at VA Research, noted that this was "a flame war," that "was no more extreme than the kind of bluster that comes from other public figures in the industry." *Id.*

317. *See, e.g.*, Brett Mendel, *Will Commercialism Help Or Hurt Linux?* (last modified Apr. 8, 1999) <http://www.cnn.com/TECH/computing/9904/08/linuxsuits.idg> (discussing the concern "that the cooperative community atmosphere for which the Linux operating system has been famous is being tainted by commercial interests."). *Cf.* Raymond, *The Revenge of the Hackers*, *supra* note 264, at 212-13. Raymond mistakenly characterizes the position of the Free Software Foundation and others as a mere tool for attaining widespread use of free software, rather than as a legitimate value choice. He argues that these "freedom" arguments should be abandoned in favor of arguments more pleasing to corporate ears. *See id.* However, causing individuals to abandon arguments that may provide the ideological underpinnings of their involvement in the community to achieve greater commercial appeal is precisely the type of tainting by commercial norms

open source community in the real world could be harmed by the conflict.[318]

Adaptation as a result of these norm clashes could occur in a number of ways. For example, the emphasis courting commercial interests may further embolden members of the open source community who attach moral significance to free software to be more vocal about their message.[319] This could provide an opportunity to reinforce the fundamental norms upon which the community was built. The offline world could adapt as well. As it gains greater exposure to the online world, its cultural expectations will change to allow different interpretations for online behavior than that same behavior would have in the real world.[320] While the precise changes that might occur are purely speculative at this time, the adaptive nature of the complex open source method indicates that the system should successfully[321] endure future normative chal-

that should be avoided.

318. *See, e.g.*, Kahney, *supra* note 316 (discussing the concern that the vocal dispute between Eric S. Raymond and Bruce Perens could negatively impact corporate America's view of open source software).

319. An example of this may be seen in the fanaticism with which Richard Stallman has begun to pursue the issue of the appropriate name of the program commonly known as "Linux," to be "corrected" to "GNU/Linux." *See* Richard Stallman, *Richard Stallman—Re: 15 Years of Free Software* (last modified Mar. 25, 1999) <http://linuxtoday.com/stories/4377.html>; Charles Babcock, *Open Source, Closed Minds* (last modified Mar. 25, 1999) <http://www.zdnet.com/zdnn/stories/comment/0,5859,2231706,00.html>. This allows him, indirectly, to raise the importance of the "freedom" aspects of free software. *See Editorial: The "Linux" vs. "GNU/Linux" debate* (last modified Apr. 8, 1999) <http://www.kt.opensrc.org/kt19990408_13.html#editorial>.

320. For example "ethical" hacking (or, as open source programmers would say "cracking") seems to have gained some approval, *see* Jim Kerstetter, *A Reprieve for 'Ethical Hacking'* (last modified July 20, 1998) <http://www.zdnet.com/zdnn/stories/news/0,4586,337644,00.html> (in legislation to update the copyright laws for the Internet cracking for research purposes would be permitted). Thus, other types of behavior that is generally condemned by society may come to be accepted in the online context, based on the difference in online norms.

321. This may depend upon how "success" is defined, however. If it is defined as everyone, everywhere using and selling nothing but open source software, this may be hindered by norms clashes. For example, the real world impression of conflict in the open source community may limit its acceptance in this community, at least in the short term, due to concerns about the ability of the community to continue into the future to develop and support open source software. However, even in this scenario the open source movement would continue, if only on a smaller scale, and could be expected to

lenges.

C. *Market Competition*

As the open source marketplace has developed, and companies have begun to make money on open source software, fears have begun to arise regarding competition within this market. Specifically, there are fears that if Linux can de-throne Microsoft in the operating system market (or some substantial portion thereof) Red Hat will simply hold the position once held by Microsoft.[322] These concerns seem unwarranted, due to the nature of the open source community.

The first reason why Red Hat will not be able to control the operating system market in the way that Microsoft has is due to the open source licensing practices. Consistent with the GPL under which Red Had issues Linux,[323] they are unable to restrict distribution of Linux. Thus, Red Hat cannot threaten Original Equipment Manufacturers (OEMs) like Dell or Compaq by requiring control over the desktop, for example, in exchange for continuing to supply them with Linux. OEMs could get a technically equivalent Linux operating system from any of a number of competing Linux distributors.[324] Unlike Microsoft, which is the only reliable source of Windows, in some sense, Red Hat is just one of many distributors of Linux. The OEMs could even continue to use the Red Hat Linux software they already obtained, since the GNU GPL gives them the "right to redistribute anything it wants as long as it con-

continue to produce high quality software. Its continued existence would allow for future attempts to penetrate other markets.

322. *See* Nicholas Petreley, *Linux And The Monopoly Game* (last visited Jan. 31 2000) <http://www.linuxworld.com/linuxworld/lw-1999-01/lw-01-penguin.html>.

323. *See Red Hat Linux 6.1: The Official Red Hat Linux Getting Started Guide* (last visited Feb. 6, 2000) <http://www.redhat.com/support/manuals/RHL-6.1-Manual/getting-started-guide/gpl.html>.

324. Other distributors include, in addition to Red Hat, Caldera <http://www.calderasystems.com>, Debian <http://www.debian.org>, Mandrake <http://www.linux-mandrake.com>, PowerPC Linux Project <http://www.linuxppc.org>, WorkGroup Solutions <http://www.wgs.com>, Trans-Ameritech <http://www.trans-am.com>, Apple Computer / The Open Group Research Group <http://www.mklinux.apple.com>, Walnut Creek <http://www.cdrom.com>, Stampede <http://www.stampede.org>, S.u.S.E. Linux <http://www.suse.com>, Pacific Hi-Tech <http://www.pht.com>, Yggdrasil Computing, Inc. <Http://www.yggdrasil.com>.

tinues to make the source code available."[325]

However, Red Hat is not *just* one of many distributors of Linux—it is arguably the most prominent.[326] Red Had could require OEMs to stop using the "Red Hat" name with the products that are distributed. Thus, Red Hat could theoretically leverage its brand equity to try to engage in anticompetitive activities. However, it is likely that this would only be a viable threat on the smaller OEMs. Large OEMs, like Dell and Compaq, have substantial brand equity themselves. And in a relative sense, Red Hat could lose more from not being distributed with one of these prominent OEMs than the OEMs would lose from not being able to use the Red Hat name.[327] Thus, the GPL, by leaving Red Hat with only its brand equity to leverage, seems to provide fairly strong protection against anticompetitive behavior.

To generalize from the specific example of Red Hat Linux, in order to take advantage of the benefits that the complexity of open source software brings to the technical quality of code, businesses must adhere to an open source license.[328] However, this forces these businesses to put themselves on equal footing, technologically, with all their competitors. Thus, even if a market, such as that for operating systems, is a natural monopoly, any monopoly benefits would accrue to Linux generally, rather than to any specific Linux vendor. Because multiple vendors can offer equivalent technologies, competition in other areas, such as customer support, can occur.[329] Flows in the system should tend to make not only technological improvement spread throughout the system, but economic improvement as well. Thus, increases in the number of users of Linux generally will flow (although perhaps not in equal amounts) to the various vendors of Linux, rather than to one particular company.[330]

325. Petreley, *supra* note 322.

326. In a recent poll by LinuxWorld, with a reader sample size of almost 900 indicated that people, 74% of those polled said Red Hat Linux is becoming synonymous with Linux. *See* Petreley, *supra* note 322.

327. *See id.*

328. *See* discussion *supra* Section III.

329. This is true because, unlike Microsoft, there are multiple reliable sources for Linux. *See supra* note 325 and accompanying text.

330. *See, e.g.*, Young, *Giving It Away*, *supra* note 75, at 116-17.

The flow of technology to all vendors can further facilitate exploitation of a larger segment of the potential market through price discrimination. Customers can get the basic Linux operating system technology (presently) for as little as $0.[331] Customers able to pay more can purchase essentially the same basic technology, but with greater support, installation or other benefits. Thus, competition can occur not only in the market generally, but between vendors targeting the various economic segments of the market.

D. *Civil Liability*

The issue of civil liability could be important to the future of open source software. Obviously, since many programmers are giving the software away for free, the potential costs of defending a lawsuit and paying a judgment would be prohibitive.[332] This could theoretically discourage participation in the open source process substantially. [333] Specifically, threat of lawsuits based on contract or tort theories of recovery exist that could be problematic for open source software.

1. Contract

Relevant to open source software, the U.C.C. implies warranties of merchantability[334] and fitness for a particular purpose[335]

331. For example, by downloading versions of Linux from <ftp://sunsite.unc.edu/pub/Linux/> (last visited Jan. 31, 2000)

332. *See* Perens, *supra* note 101, at 181.

333. *See id.*

334. *See* U.C.C. § 2-314 (1997). Where the seller is considered a merchant, there is an implied warranty that, in relevant part, the goods must (1) "pass without objection in the trade under the contract description," and (2) be "fit for the ordinary purpose for which such goods are used." *Id.* The U.C.C. defines a "merchant" as "a person who deals in goods of the kind or otherwise by his occupation holds himself out as having knowledge or skill peculiar to the practices or goods involved in the transaction or to whom such knowledge or skill may be attributed by his employment of an agent or broker or other intermediary who by his occupation holds himself out as having such knowledge or skill." U.C.C. § 2-104(1). The breach of implied warranty of merchantability may apply to an original manufacturer, even if the buyer purchased the product from an intervening party. *See* Pawelec v. Digitcom, Inc., 471 A.2d 60 (N.J. Super App. Div. 1984); *but see* Professional Lens Plan, Inc. v. Polaris Leasing Corp. 675 P.2d 887, 898-99 (Kan. Ct. App. 1984) *aff'd*, 710 P.2d 1297 (Kan. 1985) (privity of contract required).

335. *See* U.C.C. § 2-315 (1997). This warranty arises when, "at the time of con-

into contracts.[336] Liability for breach of implied warranties occur if the goods do not meet the standards of these warranties.[337]

The current approach taken by open source licenses is to include a term disclaiming all warranties. [338] This approach would appear to provide a reasonable likelihood of success in disclaiming implied warranties. Disclaimers of implied warranties will be upheld where they are conspicuous[339] and nonambiguous.[340] How-

tracting [the seller] has reason to know any particular purpose for which the goods are required and that the buyer is relying on the seller's skill or judgment." *Id.* The warranty requires that the goods will be suitable for this known purpose. *See id*.

336. Whether software is a "good," and thus falls under the U.C.C. has been itself a somewhat ambiguous question, although several courts have found that it is. *See, e.g.*, Communications Groups, Inc. v. Warner Communications, Inc., 138 Misc.2d 80, 527 N.Y.S.2d 341 (N.Y. Civ. Ct. 1988) (software system and equipment designed for a customer's specific needs is a good under the U.C.C.); Analysts International Corp. v. Recycled Paper Products, Inc., 1 CCH Computer Cases ¶ 45,050 (N.D. Ill. 1987) (an agreement to sell a custom-designed software system was held to be a contract for a sale of goods); RRX Industries v. Lab-Con, Inc., 772 F.2d 543 (9th Cir. 1985) (although training, repair and upgrades were part of the contract for software, it was a contract for a good). The UCITA would specifically incorporate implied warranties of merchantability and fitness for software. *See* UCITA § 403 (merchantability), § 405 (fitness for particular purpose). Since the UCITA has not yet been adopted by any states, discussions here will focus primarily on the related provisions of existing Article 2 when possible.

337. *See, e.g.,* Nielson Bus. Equip. Center, Inc. v. Montelone, 524 A.2d 1172, 1175 (Del. 1987) (defendant liable for failure to meet warranties of fitness and merchantability under lease of computer hardware, software and services).

338. *See* Perens, *supra* note 101, at 181.

339. Conspicuous means written so that "a reasonable person against whom it is to operate ought to have notice of it," U.C.C. § 1-201(10) (1999).

340. *See* MICHAEL D. SCOTT, SCOTT ON COMPUTER LAW, Vol. 2 (2d ed. 1998 Supp.) at 7-40.1 n. 162, 7-40.2 n. 168 and cases cited therein. For the relevant provisions of the UCITA see UCITA § 406. The requirements that would be imposed on open source licenses under the UCITA could be milder than those under the U.C.C. due to a more favorable definition of "conspicuous." While also defined as written so that "a reasonable person against which it is to operate ought to have noticed it," the intent of this phrase is more easily satisfied under the UCITA. A term will satisfy this definition if it is:

> (i) a heading in capitals in a size equal to or greater than, or in contrasting type, font, or color to, the surrounding text;
>
> (ii) language in the body of a record or display in larger or other contrasting type, font, or color or set off from the surrounding text by symbols or other marks that call attention to the language; and
>
> (iii) a term prominently referenced in an electronic record or display which is readily accessible and reviewable from the record or display. . .

UCITA 102(a)(15).

ever, a disclaimer has been found unconscionable[341] in such potentially relevant situations as an adhesionary contract,[342] or where a complex technology is involved, and the buyer has little knowledge, making them dependant upon the seller to determine suitability. Further, resellers of a product may not be protected by disclaimers by the developers.[343]

2. Tort

Software developers, including those in the open source community, could also be liable for negligent[344] or strict liability[345] torts.[346] Several factors may prevent tort liability for software. First, many states follow the economic loss rule, disallowing damages that are purely economic, such as loss of data.[347] Second, strict products liability generally does not apply to information.[348]

341. *See* A&M Produce Co. v. FMC Corp., 186 Cal. Rptr. 114, 123-26 (Cal. Ct. App. 1982) (listing situations where disclaimers are unconscionable).

342. Like shrinkwrap contracts, open source contracts could be seen as adhesionary, and thus not entitled to enforcement with regard to unconscionable terms. *Compare* Harper Tax Services, Inc. v. Quick Tax Ltd, 686 F. Supp. 109 (D. Md. 1988) (enforcing limitations in standard from contract for tax preparation software despite finding that contract was adhesionary) *with* Vault Corp. V. Quaid Software Ltd., 655 F. Supp. 750 (E.D. La. 1987) *aff'd* 847 F.2d 255 (5th Cir. 1988) (shrinkwrap from license not enforceable because it is a contract of adhesion).

343. For example, in *Barazzotto v. Intelligent Systems, Inc.*, 1 Computer Cas. (CCH) ¶ 45,031 at 60,270-71 (Ohio App. 1987) the Ohio court of appeals held that a shrink wrap license on the package of software disclaiming all warranties alone did not protect resellers of that software.

344. To recover for negligence, a plaintiff must show: (1) duty, (2) the breach of that duty, (3) causation, and (4) damages. *See* Restatement (Third) of Torts: Products Liability 1 (Draft 1997).

345. To recover under strict (products) liability, a plaintiff must show: (1) the product had a defect when sold or leased to the customer, (2) that the plaintiff used the product in a normal, intended or reasonably foreseeable way, and (3) that the defect was the proximate cause of the injury. *See* Restatement (Second) of Torts 402A.

346. This potential liability has become clearer in the context of potential Y2K problems. *See, e.g.* David Bender & Adam Gahtan, *Legal Aspects Of The "Year 2000 Problem,"* 532 PLI/PAT 47 (1998).

347. *See, e.g.*, Scott, *supra* note 340, at 15-10 n.34 (listing cases in Maryland, Florida, Colorado and California allowing purely economic losses, and cases in Minnesota, Wisconsin, Illinois, Pennsylvania, Florida and California disallowing economic losses).

348. *See* Winter v. G.P. Putnam's Sons, 938 F.2d 1033, 1039 (9th Cir. 1991) (plaintiffs picked mushrooms based on information in book published by defendants, and got sick. Court held that the ideas and expressions contained within a book are not a product

3. Potential Resolutions

While there are technically numerous ways that this situation could be resolved, the solutions fall within two main categories. First, the law could tend to impose civil liability on the open source project leaders and distributors. Alternatively, civil liability could generally be held to be unavailable for disclaimed warranties or torts. However, all either of these amount to is an allocation of risk with regard to problems from open source software.

If, as under the first scenario, the risk is substantially or largely imposed on the open source community, the community could be expected to adapt in a manner that would allow it to bear this burden. The primary result of this adaptation could be expected to be a shift in resources and distribution away from the free and inexpensive Internet-distributed versions of software and toward for-profit businesses such as Red Hat and Cygnus. The threat of lawsuit may deter some potential project leaders due to the potential economic hardship it could cause. This would mean that development, when it occurs, would be led by open source businesses who could bear the burden of lawsuits. It could also lead to non-profit or other organizations that would exist to facilitate open source development.[349] The likely judgment-proof nature of individuals distributing software for free may lead customers[350] to obtain software from vendors that have the resources to pay a civil judgment. Thus, the organizational structure of the open source community as a whole could adapt to allow the continuance of the open source community despite the threat of liability. However, the extent of that change cannot be reasonably determined at this time. If the potential number of users that would avail themselves of this liability is substantial, the changes will be more prevalent than if the number of users who would do so is more limited.

Without the threat of liability, as under the second category, lit-

within the scope of products liability). By analogy, software, and particularly source code should not be a "product" for these purposes either.

349. For example, an organization could provide an umbrella under which open source projects could occur, with only such money sought from software as would be necessary to provide insurance in the case of lawsuits.

350. This would likely be limited to corporate customers, who could have substantial losses due to software problems.

tle change in the open source community would likely be necessitated. The market, rather than the law, may nonetheless lead to some changes. In particular, customers that would prefer not to bear the risk of loss themselves may create demand for vendors who, for example, do not disclaim the implied warranties. The magnitude of this demand will determine the scope of changes that occur as open source vendor policy may change to compete for these customers. However, the customers may also adapt, by choosing the potentially more efficient option of insuring against the potential loss due to software problems.[351] Either way, the open source methodology can be expected to adapt and remain.

CONCLUSION

The success of open source software as an alternative to traditional economic intellectual property incentives result from the complex nature of the open source development model. However, the only way to ensure that this highly technical achievement continues is to ensure the continued complexity of the open source model. While this may provide a general guideline for others to attempt to replicate this incentive structure in similar contexts, it also highlights the fact that specific activities can pose a threat to the beneficial complexity. Ultimately, the open source community and those wishing to adopt its methods must embrace and encourage the complexity of their situations. If this is not done, at best, substantial effort will be wasted; and at worst, the system could be pushed out of its complex state, with resulting loss of product quality and adaptability.

351. Insurance against loss may be, in many cases, preferable to utilizing the courts once losses have occurred. This is arguably due to the administrative costs associated with using the legal system. *See, e.g.*, Stephen D. Sugarman, *Doing Away with Tort Law*, 73 CAL. L. REV. 558 (1985); STEVEN SHAVELL, ACCIDENTS, LIABILITY, AND INSURANCE (1979).