

LEGAL IMPLICATIONS OF OPEN-SOURCE SOFTWARE

David McGowan*

The proliferation of computer technology and the advent of the Internet have created many new relationships and problems that raise questions about traditional legal and economic principles. The development of “open-source” or “free” software is an example of this phenomenon. Unlike the traditional producers of computer software—Microsoft, for example—open-source software is often developed by computer programmers from all over the world, each submitting contributions to the code, and distributed without charge or for a minimal fee. Open-source software is generally passed from programmer to programmer, with the understanding that improvements may be made, but that the improvements must be distributed freely, without any attempt to “privatize” the program.

The existence of such relationships among programmers raise several interesting questions. First, how do large-scale open-source projects come into being? One open-source project, the GNU/Linux operating system, even threatens the market share of Microsoft’s operating systems—a feat that calls traditional economic theories on the operation of the firm into question. A more important question is whether the open-source model is sustainable and generalizable. Ultimately, one wonders what role the law will play in the open-source community—a community that seems to operate in the absence of traditional legal principles.

In this article, which was introduced at the University of Illinois College of Law Symposium on Intellectual Property Challenges in the Next Century, Professor McGowan addresses these questions. Using the GNU/Linux operating system as a case study, he probes the organization of the open-source community and the philosophies of its leading members in order to understand how traditional firm models, intellectual property, and contract law might apply. Professor McGowan concludes by review-

* Associate Professor of Law, University of Minnesota Law School.

My thanks to my friends and colleagues Jim Chen, Dan Farber, Michael Froomkin, Dan Gifford, Mark Lemley, Brett McDonnell, Miranda McGowan, Leo Raskind, and Lars Stole for their comments and suggestions. Mistakes that remain are my fault.

ing recent attempts by courts to impose traditional principles in computer software transaction disputes. Ultimately, it appears that the open-source community cannot be neatly categorized. Although many traditional firm theories—such as the formation of a hierarchy—and legal principles—such as copyright—do apply to the open-source model, these theories and principles are employed in creative ways not previously envisioned.

I. INTRODUCTION

The production and distribution of open-source or “free” software presents an interesting case study of the relationships among the theory of the firm, intellectual property rights, and contract law.¹ Firms such as Red Hat, Inc. have built businesses successful enough to win the approval of financial markets and maintain large market capitalizations even though they did not develop the software on which their work is based, do not employ the programmers who created and maintain it, do not control its future development, and cannot control any improvements they themselves make to it. One might explain all of this with a standard model in which firms make money on service, or by arguing that the capital markets for Internet firms have been irrational. But even if such explanations are true they tell only a part of the story. They do not explain how the operating system was produced in the first place.

The open-source program behind Red Hat’s business model is an operating system that is a variant of Unix. The program consists of a kernel of Linux code and a shell of code developed by the GNU Project, which together I will refer to as the GNU/Linux operating system (OS). This operating system is not “owned” by anyone if we use that term in its conventional sense—to refer to the exclusion of consumers from using the owner’s code as a means of inducing payment for use. The components of the GNU/Linux OS are copyrighted, and the rights are held by identifiable persons and firms. Under the open-source model of software production, however, these rights are used to enforce norms of the open-source community: Code may be freely copied, modified, and distributed, but only if the modifications (derivative works) are distributed on these terms as well.

1. The term “open-source” software is itself a matter of some dispute. Some members of the programming community that has given us the GNU/Linux operating system—in particular those more associated with the Free Software Foundation and the GNU project than with the Linux kernel—favor the term “free software.” (“Free” in this context refers to the freedom to copy and modify source code, not to price; GNU is a recursive acronym for “GNU’s Not UNIX”). At least one prominent member of the GNU project, Richard Stallman, believes the term “free” software conveys an ethical message that “open-source” software lacks. I discuss these issues in part III. Because the firm whose registration statements I use here—Red Hat, Inc.—uses the term “open-source software,” I generally use that term as well. I use the term “free software” when discussing the ideas of community members, such as Stallman, who favor that term.

These restrictions are imposed in form agreements such as the GNU General Public License (GNU GPL). The licenses, and the GNU GPL in particular, represent an elegant use of contractual terms and property rights to create social conditions in which software is produced on a model of openness rather than exclusion. The licenses have not been tested in court, and some analysts question whether they create legally binding rights and obligations. It is therefore all the more remarkable that the social structure these licenses support has produced valuable software that is important to the operation of the Internet and, in the case of the GNU/Linux OS, presents a credible source of future competition for Microsoft's operating systems.²

The development of the GNU/Linux OS presents an interesting case for exploring the theory of the firm. Following Coase, we often think of firms as institutions in which activity is arranged by authoritative command rather than negotiation.³ On this view, the firm consists of various inputs controlled by an entrepreneur;⁴ the scope of the firm is determined by the cost of in-house production compared to the cost of a market transaction with another firm.⁵ Viewing firms as areas of market-like activity organized hierarchically to minimize costs allows us to focus on persons within firms as individual economic agents. A rational actor assumption leads to the fear that these agents will try to do what is best for themselves rather than the firm—shirking work, appropriating opportunities, and the like.

Following Jensen and Meckling, we have given the name “agency costs” to the sum of the costs of agent misbehavior plus costs incurred to discover and stop such misbehavior.⁶ One implication of seeing the world in agency-cost terms is that we should expect firms to use various devices to align the interests of agents and firms, and such devices are in fact common. Monitoring agents is expensive for firms; that they are willing to incur such costs suggests that agent opportunism is a real and significant concern. Yet large numbers of programmers have spent time working to develop the GNU/Linux OS, and they generally have not been paid for this work. If conventional technology firms find it necessary to pay salaries and grant options to hire bright programmers who will work long hours designing and coding software, how is it that the GNU/Linux OS has become a commercially viable product that can sustain firms such as Red Hat?

2. Craig S. Smith, *Fearing Control by Microsoft, China Backs the Linux System*, N.Y. TIMES, July 8, 2000, at A1.

3. See generally R. H. COASE, *The Nature of the Firm*, in THE FIRM, THE MARKET, AND THE LAW 33, 33 (1988).

4. See *id.* at 40–42.

5. See *id.* at 42–44.

6. See generally Michael C. Jensen & William H. Meckling, *Theory of the Firm: Managerial Behavior, Agency Costs and Ownership Structure*, 3 J. FIN. ECON. 305, 308–10 (1976).

These questions bring us quickly to property rights. The standard explanation for copyright is that it is a reward system in which creators are given a right to exclude others from their work, which allows them to earn returns by charging a fee for access.⁷ The reward model sees the revenue creators earn as necessary to induce them to create the work in the first place. The cost of excluding the public from use is tolerated on the assumption that it succeeds in inducing creation that would not otherwise occur.⁸ The optimal copyright system is therefore one in which the marginal gain in creativity induced by the rights exceeds the marginal cost of exclusion plus the cost of administering the system.⁹ The production of commercially viable software under a regime of free copying, modification, and distribution therefore deserves attention. We should study the case of open-source software production to see whether our existing legal rules could produce socially desirable results at a lower cost.

This case study seeks to answer these questions: How is it that the GNU/Linux OS was produced in the first place? Is the model in which this operating system was produced sustainable? Is it generalizable? What role, if any, does the law play in making such production possible? My answers, in brief, are as follows.

Open-source software production is not about the absence or irrelevance of intellectual property rights. Open-source production instead rests on the elegant use of contractual terms to deploy those rights in a way that creates a social space devoted to producing freely available and modifiable code. In open-source production, property rights are held in reserve to discipline possible violations of community norms. Open-source production therefore does not take place in a true commons, though the low cost of copying and using code combined with the broad grants of the relevant licenses creates a situation that resembles a commons in some respects.

Though individuals or firms may charge for open-source code, the ability of community members to copy and distribute the code severely constrains pricing for the code itself. In many cases, persons who wish to obtain the code without charge may do so. The right to exclude is not actively employed to secure payment for code. Open-source works may therefore be more widely and cheaply available than code produced in a conventional, exclusion-based model.

If other variables—such as the rate of innovation and production—can be held equal, then the social cost associated with open-source production will be lower than the cost associated with conventional production. Whether the rate of innovation and production in general can be

7. See Robert A. Kreiss, *Accessibility and Commercialization in Copyright Theory*, 43 UCLA L. REV. 1, 14–16 (1995).

8. See *id.* at 8.

9. See William M. Landes & Richard A. Posner, *An Economic Analysis of Copyright Law*, 18 J. LEGAL STUD. 325, 326 (1989).

held equal is an open question. In some cases open-source production might be slower than conventional production, while in others it might be faster. Rates of development might also vary within projects, with core technical problems being solved faster than problems related to making a program easier for nonexpert consumers to use.

The licenses that enforce the property rights on which this structure rests are important to its success. The licenses provide a mechanism for enforcing norms, for distinguishing the open-source community from conventional software production and, in some cases, for providing incentives to programmers who require them. Although the agreements that define open-source code are sometimes said to create *de facto* property rights or “covenants running with the code,” these agreements in fact create a nonexclusive permission to use the code subject to certain conditions. The relevant property right is copyright, which does run with the code, which is why the permissions granted by the licenses must run with the code as well.

“Open-source” or “free” software refers to a type of license and not to the economic characteristics of particular projects. These terms encompass projects involving a million lines of code and projects involving only a thousand lines of code. The Linux kernel and a network printer patch might both be “open-source” software, but that fact shows only that such labels do little analytical work on their own.

The social structures necessary to support production of large, complex projects are different from the structures, if any, necessary to support small projects. It is a mistake to speak generally of the incentives or motivations behind “open-source” or “free” software production. The costs of coordination and working on particular projects must be taken into account. Large, complex projects have to provide some form of reward to induce programmers to contribute their work while accepting the hierarchy necessary to coordinate production. Smaller, less complex projects need not incur coordination costs, and therefore have no need for such payoffs. Programmers may receive reputational payoffs from working on such projects, but they also may work on such projects for personal reasons, such as the sheer enjoyment of programming.

The importance of intellectual property rights and open-source licenses increases as the complexity of projects and the cost of coordinating them increases. Where significant costs have to be sunk to ensure success, more robust legal and social structures are required. Where no such costs are needed, production may succeed in more anarchic structures.

In part II of this essay, I briefly discuss the transaction cost theory of the firm and the agency cost concept, as well as the reward structure of intellectual property rights. These points are quite familiar, so I do not dwell on them. I also briefly examine the registration statement of Red Hat. In part III, I examine the evolution of open-source software and its

licenses. In part IV, I integrate these two parts and consider some implications that this discussion has for the theory of the firm, intellectual property rights, and contract law.

II. SOME BASIC CONSIDERATIONS OF COST AND PROPERTY RIGHTS

“[T]he distinguishing mark of the firm is the supersession of the price mechanism.”¹⁰

—*R. H. Coase*

A. *Firms*

Several strands of economic analysis contribute to the theory of the firm.¹¹ One of the earliest and most important of these is Ronald Coase’s *The Nature of the Firm*. Coase asks what distinguishes firms from other forms of economic activity. His answer rests on his concept of firms as areas of market-like activity that are organized hierarchically rather than through arms-length transactions. For Coase, the firm is defined by a series of relationships between an entrepreneur and various factors of production, some of which are agents.¹² The entrepreneur directs the activities of the firm, and the agents agree to be directed by the entrepreneur rather than negotiating each action they perform.¹³

Coase also recognizes that markets are expensive to use and, therefore, market transactions are costly.¹⁴ From these points he concludes that the costs of market transactions lead to the emergence of firms: Firms are formed when hierarchical production is cheaper than market transactions.¹⁵ It follows that a firm’s scope—what it produces as opposed to what it buys—is determined by the difference between the cost of control and coordination within the firm and the cost of contracting in the market. Competition forces entrepreneurs and factors of production to minimize the costs of their arrangements or be displaced by firms that arrange themselves more efficiently, perhaps by opting for cheaper market transactions in producing their goods and services.¹⁶

Coase’s conception of the firm as an alternative to the market directed economists’ attention to the interactions among the constituents

10. COASE, *supra* note 3, at 33, 36.

11. For a brief and accessible summary, see generally Joseph E. Stiglitz, *Symposium on Organizations and Economics*, J. ECON. PERSP., Spring 1991, at 15. The major work is, of course, COASE, *supra* note 3. For a good account of the influence of this work, see generally Jason Scott Johnston, *The Influence of The Nature of the Firm on the Theory of Corporate Law*, 18 J. CORP. L. 213 (1993).

12. See COASE, *supra* note 3, at 39.

13. See *id.* at 38–39.

14. See *id.*

15. RONALD H. COASE, *The Institutional Structure of Production*, in *ESSAYS ON ECONOMISTS AND ECONOMICS* 9 (1994) [hereinafter COASE, *Institutional Structure*].

16. E.g., OLIVER E. WILLIAMSON, *THE MECHANISMS OF GOVERNANCE* 233–34 (1996) [hereinafter WILLIAMSON, *MECHANISMS*].

of a firm, typically focusing on equity investors, officers and directors, and employees. Alchian and Demsetz take the point quite far, arguing that entrepreneurs do not in fact exercise fiat power.¹⁷ They argue instead that a firm's employees could terminate their contracts in much the same way as a supplier could, if not more easily. The resulting model is one of a team production process organized through relationships with a central contracting agent.¹⁸ (This argument understates an employer's fiat power over employees, a point that I will examine in more detail in part IV.A.)¹⁹

Jensen and Meckling extended this work in an important paper that characterized firms as a nexus of contractual relationships, each at risk from agent misbehavior and each therefore presenting the problem of agency costs.²⁰ If agents try to benefit themselves, they may shirk or avoid work, perform it shoddily, take opportunities that should be given to the firm, favor stability and job security over profit, and so on.²¹ To minimize these costs, firms may resort to various monitoring devices to measure the performance of their agents and to put a stop to any misconduct. The behavior of agents is also subject to some competitive restraints, such as labor and (more diffusely) product markets, and the market for control of the firm.²² And as Judge Easterbrook and Dean Fischel have emphasized, legal structures play a role in minimizing costs as well.²³

In addition to competition and monitoring, firms often employ agreements designed to align the interests of agents to those of principals—in this model, the firm and its equity investors.²⁴ Stock options are examples of such agreements, as are performance incentives such as bonuses based on meeting sales or net income goals. Such measures have their own costs, including the risk that they may themselves induce agents to misbehave, such as by misstating facts to increase share price or

17. See Armen A. Alchian & Harold Demsetz, *Production, Information Costs, and Economic Organization*, 62 AM. ECON. REV. 777, 777 (1972).

18. *Id.* at 783.

19. See Oliver E. Williamson, *Visible and Invisible Governance*, 84 AM. ECON. REV. 323, 325 (1994).

20. Jensen & Meckling, *supra* note 6, at 323.

21. For an overview with an emphasis on the relationship between motivation and efficiency, see Herbert A. Simon, *Organizations and Markets*, J. ECON. PERSP., Spring 1991, at 25, 29–30.

22. E.g., Michael C. Jensen, *Agency Costs of Free Cash Flow, Corporate Finance, and Takeovers*, 76 AM. ECON. REV. 323, 323 (1986). For a concise and accessible review of these points, see Henry Hansmann, *THE OWNERSHIP OF ENTERPRISE* 11–52 (1996). For a review of transaction cost and agency principles in connection with an array of legal issues, see FRANK H. EASTERBROOK & DANIEL R. FISCHEL, *THE ECONOMIC STRUCTURE OF CORPORATE LAW* 113–17 (1991). This is an important book for tracking the legal effect of many of the insights discussed in this section. For a brief overview of Easterbrook and Fischel's contributions, see Ian Ayres, *Making a Difference: The Contractual Contributions of Easterbrook and Fischel*, 59 U. CHI. L. REV. 1391 (1992).

23. EASTERBROOK & FISCHEL, *supra* note 22, at 14–15.

24. E.g., *id.* at 9–11; David E.M. Sappington, *Incentives in Principal-Agent Relationships*, J. ECON. PERSP., Spring 1991, at 45, 47.

claiming unearned revenues to obtain a bonus. These costs become a part of the overall cost of assuring the faithful performance of agents.²⁵

Indeed, we may even think counterintuitively of the basic salary of an employee as a way of aligning incentives, insuring faithful performance of a task in return for wages. (Piece-work is an obvious example of a wage structure designed to reduce agency costs.) We do not generally think of wages this way, however, because we think of them as minimal payments required to induce employees to work at all. By working for one firm, employees forgo the chance to work at another, and the opportunity forgone is a cost that has to be covered. Getting agents to work faithfully requires something more than covering this basic cost.

Once the firm is seen as a collection of contractual relations among persons and other inputs, questions of negotiating strategy, information, opportunism, and enforcement become important. The problem has been stated as one involving incompletely specified contracts, and important analyses have examined that question in the context of vertical integration and the risk that parties might take advantage of unforeseen (and therefore unspecified) events to hold each other up.²⁶

B. Exchange

In his address upon receiving the Nobel Memorial Prize in economics, Coase pointed out that “what are traded on the market are not, as is often supposed by economists, physical entities but the rights to perform certain actions, and the rights which individuals possess are established by the legal system.”²⁷ For transactions involving intellectual property, the point certainly holds true. However structured (as a sale or license), the conventional exchange involves trading money for the right to use

25. Some measures are not costly in this way, and may benefit the firm by sending signals to capital markets. Debt has been treated this way in several papers. For a summary, see WILLIAMSON, *MECHANISMS*, *supra* note 16, at 182–83.

26. See Sanford J. Grossman & Oliver D. Hart, *The Costs and Benefits of Ownership: A Theory of Vertical and Lateral Integration*, 94 J. POL. ECON. 691, 693–95 (1986). See generally Oliver D. Hart, *Incomplete Contracts and the Theory of the Firm*, in *THE NATURE OF THE FIRM: ORIGINS, EVOLUTION, AND DEVELOPMENT* 138 (1991) [hereinafter *ORIGINS, EVOLUTION & DEVELOPMENT*]. In an earlier article, Klein, Crawford, and Alchian developed a model based on transaction costs in which asset specificity suggested ownership rather than contracting due to transaction costs associated with opportunism. See Benjamin Klein et al., *Vertical Integration, Appropriable Rents and the Competitive Contracting Process*, 21 J.L. & ECON. 297, 298–302 (1978). Klein discusses some of Coase’s criticisms of this model in Benjamin Klein, *Vertical Integration as Organizational Ownership: The Fisher Body-General Motors Relationship Revisited*, in *ORIGINS, EVOLUTION & DEVELOPMENT*, *supra*, at 213. For some context, see WILLIAMSON, *MECHANISMS*, *supra* note 16, at 56–57, 93–144. Much of the “theory of the firm” work on incomplete contracting assumes that contracts between agents and the firm will be enforced. Lars Stole and Jeffrey Zweibel have recently explored models of “at-will” firms in which agreements are not enforceable. See generally Lars A. Stole & Jeffrey Zweibel, *Intra-Firm Bargaining Under Non-Binding Contracts*, 63 REV. ECON. STUD. 375 (1996); Lars A. Stole & Jeffrey Zweibel, *Organizational Design and Technology Choice Under Intrafirm Bargaining*, 86 AM. ECON. REV. 195 (1996).

27. COASE, *Institutional Structure*, *supra* note 15, at 11.

code. The right itself stems from intellectual property laws or, in some cases,²⁸ contract law.

The core of property is the right to exclude.²⁹ The law grants property rights because society has decided that doing so makes us better off than we otherwise would be. With respect to intellectual property, a great deal of precedent and scholarly commentary conceives of the rights as a reward system in which society pays the rights holder to get access to the material in which rights have been granted.³⁰ (There are alternative theories for granting such rights, but I will not address those here.) To say that a reward system is desirable does not imply that any particular combination of rights, limitations, or access is optimal. Theory is of limited practical use here.³¹ We may agree with Professor Landes and Judge Posner that an optimal copyright regime is one where the social value of the marginal new work whose creation was induced by the grant of the rights exceeds the social cost of the work, plus the costs of administering the system.³² But we do not know where this point lies for any given

28. See *ProCD, Inc. v. Zeidenberg*, 86 F.3d 1447, 1455 (7th Cir. 1996).

29. *Coll. Sav. Bank, Inc. v. Fla. Prepaid Postsecondary Educ., Inc.*, 527 U.S. 666, 672 (1999) (“The hallmark of a constitutionally protected property interest is the right to exclude others.”).

30. *E.g.*, *Bonito Boats, Inc. v. Thunder Craft Boats, Inc.*, 489 U.S. 141, 150–51 (1988) (“The federal patent system thus embodies a carefully crafted bargain for encouraging the creation and disclosure of new, useful, and nonobvious advances in technology and design in return for the exclusive right to practice the invention for a period of years.”); *Harper & Row Publishers, Inc. v. Nation Enters.*, 471 U.S. 539, 546 (1985) (“The rights conferred by copyright are designed to assure contributors to the store of knowledge a fair return for their labors.”); *Sony Corp. v. Universal City Studios, Inc.*, 464 U.S. 417 (1984) (“The immediate effect of our copyright law is to secure a fair return for an ‘author’s’ creative labor. But the ultimate aim is, by this incentive, to stimulate artistic creativity for the general public good.”); *Diamond v. Chakrabarty*, 447 U.S. 303, 307 (1980) (“The patent laws promote this progress by offering inventors exclusive rights for a limited period as an incentive for their inventiveness and research efforts.”); *Twentieth Century Music Corp. v. Aiken*, 422 U.S. 151, 156 (1975) (“The immediate effect of our copyright law is to secure a fair return for an ‘author’s’ creative labor. But the ultimate aim is, by this incentive, to stimulate [the creation of useful works] for the general public good.”); *Mazer v. Stein*, 347 U.S. 201, 219 (1954) (“The economic philosophy behind the clause empowering Congress to grant patents and copyrights is the conviction that encouragement of individual effort by personal gain is the best way to advance public welfare through the talents of authors and inventors in ‘Science and useful Arts.’”); *Universal Oil Prods. Co. v. Globe Oil & Ref. Co.*, 322 U.S. 471, 484 (1944) (“As a reward for inventions and to encourage their disclosure, the United States offers a seventeen-year monopoly to an inventor who refrains from keeping his invention a trade secret.”); *Paulik v. Rizkalla*, 760 F.2d 1270, 1276 (Fed. Cir. 1985) (“The reason for the patent system is to encourage innovation and its fruits”); William W. Fisher III, *Reconstructing the Fair Use Doctrine*, 101 HARV. L. REV. 1661, 1687 (1988). Fisher states:

[T]he elaborate combination of grants and reservations that comprise the Copyright Act is designed to advance the public welfare by rewarding creative intellectual effort sufficiently to encourage talented people to engage in it, while at the same time making the fruits of their genius accessible to as many people as possible as quickly and as cheaply as possible.

Id.; Kreiss, *supra*, note 7, at 14 (“The copyright system seeks to promote the public benefit of advancing knowledge and learning by means of an incentive system. The economic rewards of the marketplace are offered to authors in order to stimulate them to produce and disseminate new works.”); Landes & Posner, *supra* note 9, at 326; Mark A. Lemley, *Romantic Authorship and the Rhetoric of Property*, 75 TEX. L. REV. 873, 889–90 (1997) (reviewing JAMES BOYLE, SHAMANS, SOFTWARE, AND SPLEENS: LAW AND THE CONSTRUCTION OF THE INFORMATION SOCIETY (1996)).

31. *E.g.*, Frank H. Easterbrook, *Cyberspace Versus Property Law?*, 4 TEX. REV. L. & POL. 103, 104 (1999) [hereinafter Easterbrook, *Property Law*].

32. See Landes & Posner, *supra* note 9, at 341–43.

product at any given time, much less for each type of product to which the rights apply.³³

For intellectual property rights to produce the revenues this model presumes necessary to sustain creation, the right to exclude must be exchanged for money. What facilitates exchange? Judge Easterbrook provides a neoclassical response: clearly defined property rights and contract rules maximizing the parties' freedom to make their own agreements.³⁴ Thus, his three principles designed to banish the law of the horse from cyberspace: "make rules clearer, to promote bargains;" "create property rights, where there are none—again to make bargains possible;" and "create bargaining institutions."³⁵

C. *A View from the Red Hat S-1*

Against this background, it is odd to see a firm selling securities to the public with the following warning:

We have not demonstrated the success of our open source business model, which gives our customers the right freely to copy and distribute our software. No other company has built a successful open source business. . . . [O]pen source vendors are not able to provide industry standard warranties and indemnities for their products, since these products have been developed largely by independent parties over whom open source vendors exercise no control or supervision.³⁶

The lack of warranties is not particularly surprising, for Red Hat would most likely disclaim them anyway.³⁷ But for Red Hat to have such an operating system to build upon is quite interesting, particularly given Red Hat's contrast of open-source software with what it calls "proprie-

33. See Easterbrook, *Property Law*, *supra* note 31, at 105.

34. Frank H. Easterbrook, *Cyberspace and the Law of the Horse*, 1996 U. CHI. LEGAL F. 207, 210–16 [hereinafter Easterbrook, *Law of the Horse*]. On the importance of exchange, see also Robert P. Merges, *The End of Friction? Property Rights and Contract in the "Newtonian" World of On-Line Commerce*, 12 BERKELEY TECH. L.J. 115, 121–28 (1997).

35. Easterbrook, *Law of the Horse*, *supra* note 34, at 209; see also Easterbrook, *Property Law*, *supra* note 31, at 111–12.

36. Red Hat, Inc., Amendment No. 5 to Form S-1, 6 (Aug. 11, 1999), available at <http://www.sec.gov/Archives/edgar/data/1087423/0001047469-99-030827.txt> [hereinafter Red Hat S-1] (on file with the *University of Illinois Law Review*). Using Red Hat's S-1 presents a risk to substantive analysis of open-source production: Red Hat may have exaggerated the risks and uncertainties of its business in order to protect itself from liability later on. There is some anecdotal evidence that technology firms issuing stock in the recent Internet boom market have spent more time describing risk factors than firms have in the past. See Terzah Ewing, *Small-Stock Focus: IPOs More Boldly Disclose Risk Factors*, WALL ST. J., Apr. 3, 2000, at C1. The risk factors I use here, however, describe real problems that pose legitimate questions for the theory of the firm and for contract and intellectual property law. Even discounted to some degree (and S-1s are selling documents as well as risk-avoiding documents), these excerpts present a fair description of the questions that need to be analyzed.

37. See Red Hat S-1, *supra* note 36, at 14–15 ("Although our license agreements with our customers typically contain provisions designed to limit our exposure to potential product liability claims, it is possible that these provisions may not be effective or enforceable under the laws of some jurisdictions."); see also UNIF. COMPUTER INFO. TRANSACTIONS ACT (UCITA) § 406 (West 2000).

tary” software: “Unlike proprietary software, open-source software has publicly available source code and can be copied, modified and distributed with minimal restrictions.”³⁸ Thus,

[U]nder the open-source software model, software is created through the collaborative efforts of large communities of independent developers. Developers work alone or in groups to write code, make the code available over the internet, solicit feedback on it from other developers, then modify and share it with others for general use. This continuous process results in the rapid evolution and improvement of open source software.³⁹

Thus far, Red Hat’s model seems a tribute to Hayekian localized knowledge and decentralization.⁴⁰ But one wonders how these programmers coordinate their work. Who decides, for example, which modifications are desirable and which are not? Who develops and maintains the code in a coherent form? On this point, with respect to the Linux kernel, the Red Hat S-1 goes on to say that:

We may not be able to release major product upgrades of Red Hat Linux on a timely basis because the heart of Red Hat Linux, the Linux kernel, is maintained by third parties. Linus Torvalds, the original developer of the Linux kernel and a small group of independent engineers are primarily responsible for the development and evolution of the Linux kernel. If this group of developers fails to further develop the Linux kernel or if Mr. Torvalds or other prominent Linux developers . . . were to join one of our competitors or no longer work on the Linux kernel, we will have to either rely on another party to further develop the Linux kernel or develop it ourselves.⁴¹

Red Hat might have a problem developing its product by itself, for its business model relies at least in part on the cost advantage of free labor from programmers who work on Linux: “Vendors are able to leverage the community of open source developers, allowing them to reduce development costs and decrease their time to market.”⁴²

The legal aspects of Red Hat’s business are as intriguing as these business elements. Let us return for a moment to the distinction Red Hat advances between proprietary and open-source software. In terms of what is exchanged, Red Hat says that:

Under the proprietary model of software development, a software developer generally licenses to the user only the object or binary code. Binary code consists of the 1s and 0s that only the computer

38. Red Hat S-1, *supra* note 36, at 2.

39. *Id.*

40. For a quick summary of this point, see Joseph Farrell, *Information and the Coase Theorem*, J. ECON. PERSP., Fall 1987, at 113, 116.

41. Red Hat S-1, *supra* note 36, at 6–7.

42. *Id.* at 2. Reducing time to market also reduces the effective cost of production, the reduction being at least the time value of money for the shorter period before investment in a product begins generating returns.

understands. By contrast, under the open source development model, the software developer provides to the user access to both the binary code and the source code. Source code is the language used by the developers. The principal differentiating points of open source software include: . . . license terms—under open source licenses, the user has access to both binary and source code, and the rights to copy, modify, alter and redistribute the software . . .⁴³

This last statement means not only that Red Hat has access to improvements made by other persons and firms, but that Red Hat must give other persons and firms access to its own improvements. These requirements are embodied in the licenses on which open-source development is based:

Red Hat Linux has been developed and made available for licensing under the GNU General Public License and similar licenses. These licenses generally permit anyone to copy, modify and distribute the software, subject only to the restriction that any resulting or derivative work is made available to the public under the same terms. Therefore, although we retain the copyrights to the code that we develop ourselves, due to the open source nature of our software products and the licenses under which we develop and distribute them, our most valuable intellectual property is our collection of trademarks.⁴⁴

And there are legal risks. The GNU GPL under which Red Hat Linux is distributed is a socially and legally complex document that, as of this date, is untested. Thus, Red Hat warns investors of

RISKS RELATED TO LEGAL UNCERTAINTY

WE COULD BE PREVENTED FROM SELLING OR DEVELOPING OUR PRODUCTS IF THE GNU GENERAL PUBLIC LICENSE AND SIMILAR LICENSES UNDER WHICH OUR PRODUCTS ARE DEVELOPED AND LICENSED ARE NOT ENFORCEABLE. The Linux kernel and the Red Hat Linux operating system have been developed and licensed under the GNU General Public License and similar licenses. These licenses state that any program licensed under them may be liberally copied, modified and distributed. We know of no circumstance under which these licenses have been challenged or interpreted in court. Accordingly, it is possible that a court would hold these licenses to be unenforceable in the event that someone were to file a claim asserting proprietary rights in a program developed and distributed under them. Any ruling by a court that these licenses are not enforceable, or that Linux-based operating systems, or significant portions of them, may not be liberally copied, modi-

43. *Id.* at 40.

44. *Id.* at 53. Red Hat also states that it “share[s] all of our developments on and improvements to the Linux kernel and other open source products with the development community.” *Id.* at 42.

fied or distributed, would have the effect of preventing us from selling or developing our products.⁴⁵

These risks did not seem to dampen the market's enthusiasm for Red Hat's IPO. Red Hat priced at \$14 per share and, in a now-familiar pattern, traded as high as \$56 per share on its first day, before closing at \$52.⁴⁶ With 67 million shares outstanding, this produced an initial market capitalization of \$3.5 billion.⁴⁷

III. THE OPEN-SOURCE COMMUNITY AND ITS TENETS

"To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or ask you to surrender your rights."⁴⁸

— *The GNU General Public License*

Open-source software projects are the result of work by members of a community of developers. Community members live and work around the world, and membership is fluid. There is no formal hierarchy to the community, nor is there formal membership. The community is instead defined in substance by the members' agreement on certain core principles, by their work on projects, and by mutual recognition. On some points members of the community agree only in part, or for different reasons, and on some points they disagree. Section A of this part examines the terms of open-source licenses and, in particular, the GNU General Public License, which supports the GNU/Linux operating system to which the Red Hat S-1 referred. Section B examines some of the norms of the open-source community, as articulated by some of its more prominent members. Section C examines how legal commentators have assessed the open-source community.

A. *Open-Source and Free Software Licenses*

To say that code is "open-source" or "free" software is to say that it is subject to one of a particular category of licenses. The most important of these is the GNU General Public License.⁴⁹ To understand the legal implications of open-source production, one must first understand the license terms and their relationship to the community's work.

Some members of the open-source community have posted *The Open Source Definition*, which sets out several conditions a license must

45. *Id.* at 14.

46. Matt Richtel, *Share Price More Than Triples in Red Hat's Public Offering*, N.Y. TIMES, Aug. 12, 1999, at C3.

47. *Id.*

48. Free Software Foundation, *GNU General Public License* (version 1.7, June 1991), at <http://www.fsf.org/copyleft/gpl.html> (last modified July 31, 2000) [hereinafter *GNU General Public License*] (on file with the *University of Illinois Law Review*).

49. *Id.*

satisfy if the code subject to the license is to qualify as “open-source software.”⁵⁰ Some terms under this definition pertain to distribution of the code. Programs distributed under an open-source license “must include source code, and must allow distribution in source code as well as compiled form.”⁵¹ (Posting the source code on the Internet satisfies this requirement.)⁵² An open-source license “may not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources,” nor may the license “require a royalty or other fee for such a sale.”⁵³

The Open-Source Definition states that licenses “must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.”⁵⁴ The definition does not require that derivative works be made available on the same terms as the code a licensee received, however. Some licenses, of which the GNU GPL is one, do require that source code to derivative works be made freely available. In the terminology of the Free Software Foundation, source-code licenses that require derivative works to be “free software” (open-source) are “copyleft” licenses.⁵⁵ Source-code licenses that allow free copying but do not contain such a requirement are “free software” licenses but are not “copyleft” licenses.⁵⁶

The Open-Source Definition has some more detailed requirements as well. One of these is that licenses may not discriminate among persons, groups, or fields of endeavor.⁵⁷ Others relate to the reputations of community members as skillful programmers.⁵⁸ While a license must allow users to modify the code and make derivative works, it may require them to distribute modified source code in two parts: the original code as written by the licensor and, under a separate name or version number, the modifications of the licensee. The definition also requires that open-source licenses not “contaminate” other software distributed along with software subject to the license.

50. See Bruce Perens, *The Open Source Definition* (June 1997), at <http://www.opensource.org/osd.html> (on file with the *University of Illinois Law Review*).

51. *Id.* ¶ 2.

52. *See id.*

53. *Id.* ¶ 1.

54. *Id.* ¶ 3.

55. See Free Software Foundation, *What Is Copyleft?*, at <http://www.gnu.org/copyleft/copyleft.html> (updated Aug. 19, 2000) (on file with the *University of Illinois Law Review*).

56. *See id.*; Free Software Foundation, *What Is Free Software?*, at <http://www.gnu.org/philosophy/free-sw.html> (updated June 18, 2000) (on file with the *University of Illinois Law Review*); Free Software Foundation, *Various Licenses and Comments About Them*, at <http://www.gnu.org/philosophy/license-list.html> (updated Aug. 18, 2000) (on file with the *University of Illinois Law Review*).

57. Perens, *supra* note 50, ¶¶ 5–6.

58. Here and elsewhere in this paper, I use the term “reputation” to refer to the information available to the open-source community, and to labor markets, concerning a programmer’s competence. The term is often used in the economic literature to refer to an agent’s commitments to refrain from opportunistic behavior, which is not the point I wish to make here.

Several well-known licenses satisfy *The Open-Source Definition*. The GNU GPL is the archetype of such licenses, and is one of the stricter licenses that complies with the definition. The GPL sets out a two-pronged strategy for enforcing community norms. The first is to have the original author retain the copyright in the author's code.⁵⁹ The second is to allow others to use, modify, and redistribute the code only if they agree to comply with the GPL's terms.⁶⁰ These terms include restrictions on use that embody the community's norms. If a licensee violates the norms, the authors or their assignees may enforce them through an infringement action.⁶¹

The GNU GPL defines two working terms: "program" means a work subject to the license, and "work based on the program" refers either to the program "or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it."⁶² The basic license term provides that:

[Licensees] may copy and distribute verbatim copies of the Program's source code as [they] receive it . . . provided that [they] conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty . . . and give any other recipients of the Program a copy of this License along with the Program.⁶³

Licensees may also "modify [their] copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work" so long as they follow the same terms.⁶⁴ For derivative works, licensees must also "cause any work that [they] distribute or publish . . . to be licensed as a whole at no charge to all third parties *under the terms of this License*."⁶⁵

59. The Free Software Foundation prefers that authors assign their rights to FSF. See *infra* Part IV.C.

60. See *GNU General Public License*, *supra* note 48, preamble.

61. See *id.*

62. *Id.* ¶ 0.

63. *Id.* ¶ 1.

64. *Id.* ¶ 2.

65. *Id.* ¶ 2(b) (emphasis added). The license also states that these terms apply to "the modified work as a whole" but not to "identifiable sections of that work [that] are not derived from the Program, and [that] can be reasonably considered independent and separate works in themselves" when such independent works are distributed on their own. When independent works are distributed as part of the whole, however, they are subject to the license. *Id.* ¶ 2. The section further states that "mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or a distribution medium does not bring the other work under the [scope of this] License." *Id.* In other words, independent works integrated within a larger work derived from open-source code are subject to the GNU GPL. Independent works that are merely distributed on the same disk or CD-ROM as a work derived from open-source code are not subject to the GNU GPL.

This term seems to be the source of some confusion within the open-source community, and has led some to suggest that the GNU GPL does not comply with the noncontamination requirement of the open-source definition. See Josh Lerner & Jean Tirole, *The Simple Economics of Open Source*, 7 (Feb. 25, 2000), available at <http://www.people.hbs.edu/jlerner/simple.pdf> (on file with the *University of Illinois Law Review*). This statement is not correct, because the open-source definition requires

Under this model, “each time [licensees] redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute, or modify the Program subject to these terms and conditions.”⁶⁶ The GPL further provides that a licensee “may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt to otherwise copy, modify, sublicense or distribute the Program is void, and will automatically terminate [the] rights under this License.”⁶⁷ In this event, however, “parties who have received copies, or rights, . . . under this License will not have their licenses terminated so long as such parties” themselves comply fully with the license terms.⁶⁸

The GNU GPL also includes provisions ensuring that the code includes information about the programmers who wrote it. The license allows licensees to modify the licensed code so long as they “cause the modified files to carry prominent notices stating that [the files were] changed and the date of any change.”⁶⁹ This information allows those who see the code to form opinions about the programmer’s skill and,

only that a license not “place restrictions on other software that is distributed along with the licensed software.” Perens, *supra* note 50. The GNU GPL allows independent works to be distributed on the same media as GPL-licensed software even if the independent software is not subject to the GPL. As version 1.7 of the open-source definition recognizes, “GPLed libraries ‘contaminate’ only software to which they will be actively linked at runtime, not software with which they are merely distributed.” *Id.* That being said, Lerner and Tirole are right to note that the GNU GPL contaminates even independent works that are “bundled” with GPL-licensed code. Lerner & Tirole, *supra*, at 7.

Lerner and Tirole’s discussion of the relevant licenses is incorrect in another respect. Lerner and Tirole refer to the Debian Social Contract and the Open-Source Definition as types of licenses, which they are not. *Id.* These documents (the latter is a refinement of principles set forth in the former) set forth characteristics that licenses must satisfy to qualify as “open-source,” but the documents do not themselves grant rights or impose limitations. Lerner and Tirole link increased success of open-source projects to the use of more liberal licenses and the “concomitant decline of the GNU license.” *Id.* at 9. They discuss three open-source projects: Apache, Perl, and Sendmail. Each project has its own license, and each of these licenses is indeed less restrictive than the GNU GPL. Richard Stallman, *Various Licenses and Comments About Them*, *supra* note 57 (Apache license is a “simple, permissive non-copyleft free software license” that is not compatible with the GNU GPL; Perl license is “a free software license, but it may not be a real copyleft. It is compatible with the GNU GPL because the GNU GPL is one of the alternatives.”). Stallman does not discuss the Sendmail license, which is a free software but not a copyleft license. The GNU/Linux OS is a product of the GNU GPL, however, and the rapid growth of such a complex project under the GPL suggests the copyleft license term is viable in at least some cases.

66. *GNU General Public License*, *supra* note 48, ¶ 6. This paragraph concludes by saying that the licensee is not responsible for enforcing compliance with the license terms.

67. *Id.* ¶ 4.

68. *Id.*

69. *Id.* ¶ 2(a). The preamble to the GNU Lesser GPL connects this provision explicitly to the programmers’ reputation:

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author’s reputation will not be affected by problems that might be introduced by others.

Id.; Free Software Foundation, *GNU Lesser General Public License* (version 2.1, Feb. 1999), at <http://www.gnu.org/copyleft/lesser.html> (last updated June 18, 2000) [hereinafter *GNU Lesser General Public License*] (on file with the *University of Illinois Law Review*).

over time, allows programmers to develop reputations for expert or in-expert programming.

With respect to questions of formation, the GPL has this to say:

You are not required to accept this license, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.⁷⁰

In aid of this provision, the GNU GPL also states that licensees who modify a program that “normally reads commands interactively when run” must cause the program “when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice,” and a disclaimer of warranties.⁷¹ The announcement must also state “that users may redistribute the program under these conditions, and [tell] the user how to view a copy of this License.”⁷²

To see how these terms are designed to operate, and setting aside for the moment the question whether the model is legally viable, imagine three parties: Regan, Cordelia, and Goneril. Suppose that Regan writes a program and distributes it to Cordelia under the GNU GPL. Regan either does or does not give Cordelia enough notice of the GPL terms to form a license agreement satisfying the formation requirements of applicable contract law. If she does, then let us assume that Cordelia is bound.⁷³ If Regan does not give Cordelia enough notice to form a binding agreement, then Cordelia has no license to copy, modify, or distribute the code. If Cordelia does any of these things, Regan may sue her for infringement. All this is merely to say that licenses are defenses to infringement actions.

If Cordelia is bound, her ability to produce and distribute derivative works is constrained by section 2(b) of the GPL. If Cordelia violates this provision, her rights to create derivative works terminate and Regan may sue her for infringement.⁷⁴ Assume that Cordelia modifies the program, creating a derivative work, and sends the derivative work to Goneril. Cordelia owns the rights in her contribution to the derivative work;

70. *GNU General Public License*, *supra* note 48, ¶ 5.

71. *Id.* ¶ 2(c).

72. *Id.*

73. *E.g.*, *ProCD, Inc. v. Zeidenberg*, 86 F.3d 1447 (7th Cir. 1996).

74. The termination provision is ¶ 4. Regan’s ability to enjoin Cordelia would technically depend on whether section 2(b) is a limitation on use or a contractual promise. *Sun Microsystems, Inc. v. Microsoft Corp.*, 188 F.3d 1115, 1119 (9th Cir. 1999). Particularly given the social context in which the GPL is most commonly used, it seems clear that section 2(b) is a limitation on use and violation of its restrictions would therefore support an action for infringement as well as breach of contract.

Regan owns the rights to the original code, subject to the license to Cordelia.⁷⁵ Cordelia either does or does not give Goneril enough notice of the GPL terms to bind Goneril. If she does give notice, Goneril is bound by the GPL. If she does not, Regan may assert that Cordelia's failure to give notice violated section 1 of the GPL and thus, under section 4, terminated Cordelia's rights, including the rights to create a derivative work and to distribute Regan's original code. Regan may sue Cordelia for breach of the GPL and, possibly, for infringement.

Such a suit might be very complex. Cordelia probably could continue to distribute the code she added to Regan's, in which Cordelia holds the rights, but she could not distribute Regan's code. This is because at the time she wrote her code, Cordelia held a license from Regan to create a derivative work. Cordelia's distribution of her code in violation of section 2(b) would terminate the rights she had to distribute or modify Regan's code. But Cordelia's violation, and therefore the termination of her rights, would not occur until after she had written her code, in which she receives title at the time she writes it. Cordelia's creation of a derivative work therefore would not, in and of itself, violate Regan's rights. It is only Cordelia's subsequent distribution of her derivative work in a manner at odds with section 2(b) that would violate the GPL. (Perhaps Regan could allege that Cordelia acted in bad faith all along and, therefore, should not be considered to have had a license to create derivative works at all, but this seems a very speculative contention.) This theoretical complexity might be of little practical importance: Cordelia's code might be worth very little if anything without Regan's code.⁷⁶

As to Goneril, Regan might sue Goneril for infringement if Goneril uses the code she received from Cordelia (which includes both Cordelia's code and Regan's) in a manner inconsistent with the GPL. If Goneril adheres to the GPL terms, however, she might rely on section 4 of the GPL for a defense and a continuing right to use the code. And if Goneril has suffered any harm because of Cordelia's failure to notify her of the relevant restrictions, then Goneril might try her hand at indemnity or contribution claims against Cordelia. Either way, so long as Regan retains the original rights to her code, she should be able to enforce the open-source norms embodied in the GPL.

Section 2(b) of the GPL does not apply to Regan, who owns the full panoply of rights. This fact presents some risk that Regan might release open-source code to the community and then attempt to take the code private, revoking the GPL rights of her licensees and distributing her

75. 17 U.S.C. § 103(b) (1994); *Stewart v. Abend*, 495 U.S. 207, 223 (1990) ("The aspects of a derivative work added by the derivative author are that author's property, but the element drawn from pre-existing work remains on grant from the owner of the pre-existing work."). The GPL does not vest ownership of the derivative work in the licensor, so a court presumably would consider the author of the new code to hold the rights in that code.

76. The question of the relative importance of the licensor's code compared to the collective derivative works of the licensees is an important one, which we discuss further in part IV.C.

code in binary form and for a profit. She can unquestionably do this with respect to her own code. But for projects on which many community members have worked, the question is more complicated. Assuming that Cordelia had a license to write derivative code at the time she wrote it, then she owns the rights to her independent contributions and subsequent termination of her GPL rights should not change that fact. If Regan chooses to incorporate Cordelia's work into the original program, then as to that new code Regan is a licensee and is bound by section 2(b).⁷⁷ Under the open-source model, programs can easily become (indeed are designed to be) quilts of code from many different authors, each of whom own rights as to which the others are licensees, and each of whom uses code subject to section 2(b). This web of blocking copyrights suggests that, as a practical matter, each contributing programmer would have to agree to privatize the code if it was to be taken private in its most current and complete form.⁷⁸

The GNU project also has created the GNU Lesser Public License, which is available for and applies to only certain code libraries "in order to permit linking those libraries into non-free programs."⁷⁹ A commercial application developer wishing to use open-source code might balk at the GPL requirements, which would treat the developer's program as a derivative work if the program made use of open-source code when running. (Assuming that this portion of the GPL is enforceable, it implies that a commercial program that makes use of GPL-licensed code could itself be distributed and modified under the GPL.) Because the array of compatible programs strongly influences the value of some code—operating systems being a particular example—it makes sense for those wishing to popularize open-source programs such as the GNU/Linux operating system to give commercial developers the option of using open-source libraries without subjecting their own conventional (binaries-only, for-profit) code to open-source treatment.⁸⁰

To achieve this goal, the Lesser GPL distinguishes between programs that contain open-source library material (a "work based on the library") and those that are only designed to be compiled or linked with an open-source library (a "work that uses the library"). The former are derivative works and subject to open-source requirements; the latter, standing alone, are not.⁸¹ The Lesser GPL also provides, however, that:

77. We know this because Cordelia was required to release her derivative code under the GPL.

78. See Mark A. Lemley, *The Economics of Improvement in Intellectual Property Law*, 75 TEX. L. REV. 989, 1074–77 (1997).

79. *GNU Lesser General Public License*, *supra* note 69, preamble.

80. As the preamble says, "[f]or example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard." *Id.* For a discussion of the value added by network effects and the importance of standards, see Daniel J. Gifford & David McGowan, *A Microsoft Dialog*, 44 ANTITRUST BULL. 619 (1999); Mark A. Lemley & David McGowan, *Could Java Change Everything? The Competitive Propriety of a Proprietary Standard*, 43 ANTITRUST BULL. 715 (1998).

81. *GNU Lesser General Public License*, *supra* note 69, ¶ 5.

[A programmer] may also combine or link a “work that uses the Library” with the Library to produce a work containing portions of the Library, and distribute that work under terms of [the programmer’s] choice, provided that the terms permit modification of the work for the customer’s own use and reverse engineering for debugging such modifications.⁸²

In this event, the distributor must comply with several additional conditions.

B. *Open-Source Community Norms*

The terms “open-source” and “free” software tell us little about the code to which they apply. The GNU shell, the Linux kernel, patches for various hardware and software problems, and hacks of DVD encryption software might all be defined by these terms. The terms reveal only a portion of the varied and, in some cases, complex social structures that support the production of open-source software. Nevertheless, the terms do represent some tenets that community members hold in common. In this section, we will examine those tenets as well as some of the differing opinions within the community on the nature of the social structures that make production possible.

The conditions necessary for the production of creative works have been discussed at length in legal and economic literature. The concepts developed in these discussions are useful, but one must guard against the reflexive use of familiar concepts to explain unfamiliar behavior. We must take care not to interpret the activity we see in the way that best suits the concepts we know, but to first understand the activity and then ask whether the concepts can help us make sense of it.⁸³ For this reason, and because I am not a member of this community and cannot speak directly to its norms, I rely on the writings of three prominent community members to explain its tenets and behavior.

We begin with Richard Stallman, a founder of the Free Software Foundation (FSF) and an early force behind what he prefers to call the “free software” movement.⁸⁴ In the 1970s, Stallman worked for the Artificial Intelligence (AI) Lab at MIT.⁸⁵ Software was shared freely among lab members, and Stallman embraced the culture of sharing and openness.⁸⁶ This culture included providing source code to those who wanted it, which allowed them to improve programs or modify programs to fit

82. *Id.* ¶ 6.

83. *Cf.* GEOFFREY R. ELTON, *THE PRACTICE OF HISTORY* 51–58 (1967) (discussing difficulties in understanding history presented by the historian’s conceptual grounding in the present).

84. Richard M. Stallman, *The GNU Project* (section entitled the Free Software Foundation) (1998), at <http://www.gnu.org/gnu/the-gnu-project.html> (updated Dec. 4, 2000) [hereinafter Stallman, *The GNU Project*] (on file with the *University of Illinois Law Review*).

85. *Id.*

86. *Id.*

particular needs.⁸⁷ In 1981, many of the hackers at MIT's AI Lab left to join a private firm.⁸⁸ The lab community suffered, and in January 1984, Stallman left MIT and began writing GNU software.⁸⁹ Stallman wanted to continue this open mode of programming, and he left MIT to avoid the risk that the university would own his code.⁹⁰ He chose not to join a private firm because he objected to the closed nature of the code produced under the conventional software production model.⁹¹ Stallman helped form FSF in 1985 and has been active in the community since that time.

Stallman objects on ethical grounds to the protection of software through copyright or other legal rules that allow authors to exclude others from the code they write.⁹² For Stallman, "the law should conform to ethics, not the other way around."⁹³ As he sees it, "signing a typical software license agreement means betraying your neighbor: 'I promise to deprive my neighbor of this program so that I can have a copy for myself.'"⁹⁴ Stallman sees the exclusion-based model of the copyright reward system as inherently divisive and antisocial.⁹⁵ He views FSF work as a first step away from that model.⁹⁶ He recognizes that some trade-offs in productivity might result—"if we eliminate intellectual property as a means of encouraging people to develop software, at first less software will be developed"—but he believes the trade-offs might not be severe because free software "will be more useful" and because "there are other ways to encourage development."⁹⁷

Stallman objects to the term "open-source" software because he feels it does not convey the ethical point the term "free" software makes.⁹⁸ Stallman believes that persuading users to adopt software distributed under the GNU GPL is desirable but is only a first step in weaning users away from closed, exclusionary production models.⁹⁹ He sees

87. Stallman has written a number of papers covering a wide range of relevant topics, which may be accessed at www.gnu.org. I refer to these papers here only by their names; as of this writing each is available on the GNU website. Stallman's history of the free software movement is *The GNU Project*. See *id.* His justification of free software and objections to commercial software are explained in *Why Software Should Be Free*. See Richard Stallman, *Why Software Should Be Free* (1998), at <http://www.gnu.org/philosophy/shouldbefree.html> (updated May 31, 2000) [hereinafter Stallman, *Why Software Should Be Free*] (on file with the *University of Illinois Law Review*).

88. Stallman, *Why Software Should Be Free*, *supra* note 87.

89. Stallman, *The GNU Project*, *supra* note 84.

90. *Id.*

91. *Id.*

92. *Id.*

93. Stallman, *Why Software Should Be Free*, *supra* note 87.

94. *Id.*

95. *Id.*

96. *Id.*

97. *Id.*

98. Richard M. Stallman, *Why "Free Software" Is Better than "Open Source"*, at <http://www.gnu.org/philosophy/Free-Software-For-Freedom.html> (last modified July 30, 2000) (on file with the *University of Illinois Law Review*).

99. *Id.*

the “open-source” terminology as a concession by community members to make their efforts and code more palatable to the business community.¹⁰⁰ Stallman wishes to persuade users, including businesses, to “*value the freedom* free software gives them, for its own sake.”¹⁰¹ Thus, for Stallman, “the Free Software movement and the Open Source movement are like two political parties within our community. . . . We disagree on basic principles, but agree on most practical recommendations” and “work together on many specific problems.”¹⁰²

Some community members see Stallman’s arguments as excessive. Eric Raymond, a prominent hacker, open-source advocate, and a director of VA Linux, offers a more pragmatic view.¹⁰³ Raymond characterizes Stallman and the Free Software Foundation as zealously opposing commercial software and says the GNU GPL “expresses FSF’s zealous and anticommercial attitudes.”¹⁰⁴ Raymond says, however, that “[t]here was always a quieter, less confrontational strain in the hacker culture” that rested less on ideology than on “a group of engineering traditions founded on early open-source efforts which predated the FSF.”¹⁰⁵ Pragmatist hackers saw the GNU GPL as “an important tool” to encourage software sharing “rather than an end in itself.”¹⁰⁶ Raymond associates this view with developers involved in developing Berkeley Unix and distributing it under the BSD license.¹⁰⁷

Raymond has written the most extensive analysis explaining open-source development in economic terms. He sees the open-source community as being “most effectively understood not in conventional exchange-economy terms but as what anthropologists call a ‘gift culture’ in which members compete for status by giving things away.”¹⁰⁸ Raymond recognizes, however, that community members live and work in a world largely based on exchange; “[f]or most, the exchange game has lost its appeal but not its power to constrain. Their behavior has to make sufficient material-scarcity-economics sense to keep them in a gift-culture-supporting zone of surplus.”¹⁰⁹

100. *Id.*

101. *Id.*

102. *Id.*

103. Raymond’s papers may be accessed at www.tuxedo.org. I again refer only to the relevant names of the papers, each of which is available on Raymond’s website as of this writing.

104. Eric S. Raymond, *Homesteading the Noosphere*, § 2, at <http://www.tuxedo.org> (Apr. 1998) [hereinafter Raymond, *Homesteading the Noosphere*] (on file with the *University of Illinois Law Review*).

105. *Id.*

106. *Id.*

107. “BSD” stands for Berkeley Software Distribution. The BSD License is a free software license but not a copyleft license. The lack of a copyleft term in the BSD license may have been important to the splintering of UNIX. See Vinod Valloppillil, *Open Source Software, A (New?) Development Methodology* (“Halloween I”), at <http://www.opensource.org/halloween/halloween1.html> (last visited Sep. 19, 2000) (on file with the *University of Illinois Law Review*).

108. Eric S. Raymond, *The Magic Cauldron*, § 2, at <http://www.tuxedo.org> (June 1999) [hereinafter Raymond, *The Magic Cauldron*] (on file with the *University of Illinois Law Review*).

109. *Id.*

By referring to competition as a behavioral model and status as the goal of competition, Raymond places the open-source community in a context familiar to law and economics. The particulars of his model, however, are unusual. These aspects are most easily seen through the legal lenses of property and contract—here meaning copyright and licenses. According to Raymond:

[T]he implicit theory of the [Open-Source Definition (OSD)] (and OSD-conformant licenses such as the GPL, the BSD license, and Perl's Artistic License) is that anyone can hack anything. Nothing prevents half a dozen different people from taking any given open-source product (such as, say the Free Software Foundation's gcc C compiler), duplicating the sources, running off with them in different evolutionary directions, but all claiming to be the product.

In practice, however, such 'forking' almost never happens. Splits in major projects have been rare, and always accompanied by re-labeling and a large volume of public self-justification. It is clear that, in such cases . . . the splitters felt they were going against a fairly powerful community norm.

In fact (and in contradiction to the anyone-can-hack-anything consensus theory) the open-source culture has an elaborate but largely unadmitted set of ownership customs. These customs regulate who can modify software, the circumstances under which it can be modified, and (especially) who has the right to redistribute modified versions back to the community.¹¹⁰

Raymond therefore sees the open-source community as defining ownership functionally: "The owner(s) of a software project are those who have the exclusive right, recognized by the community at large, to *re-distribute modified versions*."¹¹¹ Under this definition, Raymond states that ownership may be acquired in any of three ways. An owner may be the person or group that began a project—announcing their intention to work on a problem, recruiting developers to work on it, and coordinating the developers' efforts. "When a project has only one maintainer since its inception and the maintainer is still active, custom does not permit even a *question* as to who owns the project."¹¹²

Ownership may also be transferred: "It is well understood in the community that project owners have a duty to pass projects to competent successors when they are no longer willing or able to invest needed time in development or maintenance work."¹¹³ This norm ensures active maintenance and development of the project, which in turn ensures that the developers' efforts will not go to waste.

110. *Id.* § 3.

111. *Id.* § 4.

112. *Id.* The Red Hat S-1 is consistent with this view of ownership norms. As noted above, it states as a risk factor that Red Hat might not be able to release product upgrades because the Linux kernel is maintained by Linus Torvalds.

113. *Id.*

Ownership may also be gained through a form of adverse possession:

The third way to acquire ownership of a project is to observe that it needs work and the owner has disappeared or lost interest. If you want to do this, it is your responsibility to make the effort to find the owner. If you don't succeed, then you may announce in a relevant place (such as a Usenet newsgroup dedicated to the application area) that the project appears to be orphaned, and that you are considering taking responsibility for it. . . .¹¹⁴

If you have gone through this process in sight of the project's user community, and there are no objections, then you may claim ownership of the orphaned project and so note in its history file. This, however, is less secure than being passed the baton, and you cannot expect to be considered fully legitimate until you have made substantial improvements in the sight of the user community.¹¹⁵

Though Raymond is clear that community norms weigh strongly against forking, open-source licenses make forking a possibility and it apparently happens on occasion. From a legal perspective, we may characterize forking as a fourth way of obtaining ownership: the hostile takeover. If a program "owner" maintains a project badly, or in a manner inconsistent with community norms, the threat of forking may act as a disciplinary device. As Raymond says in a later paper, "[w]hile forking is frowned upon and considered a last resort . . . it's considered critically important that the last resort be present in case of maintainer incompetence or defection (e.g., to a more closed license)."¹¹⁶

Raymond interprets open-source ownership norms as an expression of Lockean theories of property.¹¹⁷ Raymond's paper *Homesteading the*

114. There is also a period in which the adverse claim must be maintained:

Custom demands that you allow some time to pass before following up with an announcement that you have declared yourself the new owner. In this interval, if someone else announces that they have been actually working on the project, their claim trumps yours. It is considered good form to give public notice of your intentions more than once. More points for good form if you announce in many relevant forums (related newsgroups, mailing lists); and still more if you show patience in waiting for replies. In general, the more visible effort you make to allow the previous owner or other claimants to respond, the better your claim if no response is forthcoming.

Id.

115. *Id.*

116. Raymond, *The Magic Cauldron*, *supra* note 108, § 8. The degree to which forking may serve as a disciplinary device is a complex question. In theory, it might depend on the proportion of a program written by the licensor compared to the proportion comprised of derivative works. As noted above, section 2(b) of the GPL does not bind the original author, who therefore might attempt to remove her code from the community by terminating the GPL rights she had granted and beginning for-profit, binaries-only distribution of her work. She could not take licensee contributions private, however, because the authors of those contributions would own the rights to them. To the extent a licensor had incorporated licensee inputs back into her original work, she herself would be a licensee of the improvements and, as to that code, subject to section 2(b). A true hostile takeover would be possible only for projects in which the important functions of the code were performed by an intermingling of original licensed code and new code written by licensees and released back to the project under the GPL. In that case, the original author could not effectively take the project private and would have to justify his or her maintenance of the project in terms of community norms.

117. See Raymond, *Homesteading the Noosphere*, *supra* note 104, § 5.

Noosphere relates open-source ownership to norm-based property rights that hackers obtain by putting their labor into programming projects.¹¹⁸ Conceiving of open-source software this way produces a structure that fits quite comfortably into a rational-actor economic model:

The Lockean logic of custom suggests strongly that open-source hackers observe the customs they do in order to defend some kind of expected return from their effort. The return must be more significant than the effort of homesteading projects, the cost of maintaining version histories that document 'chain of title', and the time cost of doing public notifications and a waiting period before taking adverse possession of an orphaned project.¹¹⁹

In this model, hackers maximize their reputations for skill and the community's property norms evolve in order to secure reputational returns. Thus, the "yield" to hackers from their open-source work is "peer repute in the gift culture of hackers, with all the secondary gains and side-effects that implies."¹²⁰

Raymond's views are valuable to analysts, such as myself, who are not hackers and therefore cannot comment directly on hacker norms.¹²¹ But Raymond's is not the only voice from the hacker community on these matters. Eben Moglen, a professor of law and legal history at Columbia, the general counsel of the Free Software Foundation, and a self-proclaimed hacker from the age of thirteen, challenges the payoff structure Raymond describes. In an admirably direct article entitled *Anarchism Triumphant: Free Software and the Death of Copyright*,¹²² Professor Moglen argues that "Free software . . . is a commons: no reciprocity ritual is enacted there. A few people give away code that others sell, use, change, or borrow wholesale to lift out parts for something else."¹²³

118. *See id.* § 2.

119. *Id.* § 5.

120. *Id.* § 9. As I read him, Raymond here uses "gift" in the sense of obtaining stature by giving away valuable objects. The term is sometimes used to refer to a strategy of giving gifts to put others under obligations to oneself, which is not the norm Raymond describes.

121. I have not independently researched the status of open-source projects to verify Raymond's distillation of community norms. Nor does Raymond claim to have conducted a formal, exhaustive survey. He bases his summary of open-source ownership norms on his participation in the community: I have observed these customs in action for twenty years, going back to the pre-FSF ancient history of open-source software. They have several very interesting features. One of the most interesting is that most hackers have followed them without being fully aware of doing so. Indeed, the above may be the first conscious and reasonably complete summary ever to have been written down.

Another is that, for unconscious customs, they have been followed with remarkable (even astonishing) consistency. I have observed the evolution of literally hundreds of open-source projects, and I can still count the number of significant violations I have observed or heard about on my fingers.

Raymond, *The Magic Cauldron*, *supra* note 108, § 4.

122. Eben Moglen, *Anarchism Triumphant: Free Software and the Death of Copyright*, 4 FIRST MONDAY (Aug. 2, 1999), at <http://www.firstmonday.dk/issues.html> (on file with the *University of Illinois Law Review*).

123. *Id.* § III.

Writing for a general, technologically sophisticated audience rather than for legal academics, Moglen divides conventional legal analysts of software into two groups: the “IPdroids” and the “econodwarves.”¹²⁴ He argues that the various laws presently applicable to software are needlessly complex and rest on irrelevant distinctions between essentially similar phenomena.¹²⁵ Moglen sees IPdroids as having mastered the complexities of the regime—copyright, patent, and trade secret law—while having missed the essential similarity of the bitstreams to which the laws apply.¹²⁶ Mastery of the distinctions creates the illusion of erudition, in Moglen’s view, but makes useful analysis essentially impossible.¹²⁷ IPdroids are thus condemned “to know everything about something and nothing about anything else.”¹²⁸

For Moglen, however, the shortcomings of the IPdroid are as nothing to the lot of “the other primary protagonist of educated idiocy: the econodwarf. Like the IPdroid, the econodwarf is a species of hedgehog,”¹²⁹ “but where the droid is committed to logic over experience, the econodwarf specializes in an energetic and well-focused but entirely erroneous view of human nature.”¹³⁰ With his assault on econodwarves, Professor Moglen reaches the heart of the questions I am discussing here.¹³¹ Moglen objects to the conventional economic premise that people are motivated only by incentives without which they will not engage in productive activity like writing software code. As he puts it, “‘incentives’ is merely a metaphor, and as a metaphor to describe human creative activity it is pretty crummy.”¹³² For Moglen, references to the hacker culture as a “gift-exchange culture” serve only to remind us “that the economeretricians have so corrupted our thought processes that any form of non-market economic behavior seems equal to every other kind.”¹³³

In Professor Moglen’s view, creativity is essentially hard-wired in humans and needs only the proper social and technological structure to flourish: “It’s an emergent property of connected human minds that they create things for one another’s pleasure and to conquer their uneasy sense of being too alone.”¹³⁴ Thus he offers “Moglen’s Metaphorical Corollary to Faraday’s Law,” which analogizes human creative activity in

124. *See id.* § I.

125. *See id.*

126. *See id.*

127. *See id.*

128. *Id.* § 1.

129. Moglen here uses the term as Isaiah Berlin did, to distinguish persons who know one big thing (hedgehogs) from those who know many things (foxes). *See id.* (citing ISAIAH BERLIN, *THE HEDGEHOG AND THE FOX: AN ESSAY ON TOLSTOY’S VIEW OF HISTORY* 1 (1953)).

130. Moglen, *supra* note 122, § I. Econodwarves, it may be noted, indulge in what Moglen memorably calls “argument *ad pygmeam*.” *Id.*

131. Which, I suppose, makes me a dwarf. The reader is duly warned.

132. Moglen, *supra* note 122, § I.

133. *Id.* § III.

134. *Id.* § I.

a networked world to the effect of a magnetic field in inducing current to flow through a completed circuit.¹³⁵ In the reward model of intellectual property rights, the incentives stem from the right to exclude others from use. In Moglen's model, the right to exclude is merely "resistance" that impairs the ability of creative people to connect with one another and thereby enhance their individual and collective creativity. "Property concepts, whatever else may be wrong with them, do not enable and have in fact retarded progress."¹³⁶

Having said all that, however, Moglen also recognizes that FSF thus far has relied on the existing property rights structure adapted to the particular ends of the free software community. Moglen states that the GNU GPL is central to that community and represents the "use of intellectual property rules to create a commons in cyberspace."¹³⁷ It is the GNU GPL, and in particular section 2(b)'s requirement that derivative works be "free software" as well, that assures developers that they "will be able to run, modify and redistribute the program indefinitely, that source code will always be available, and that, unlike commercial software, its longevity cannot be limited by the contingencies of the marketplace or the decisions of future developers."¹³⁸

Professor Moglen is equally clear that the free software community has thrived in the past decade because technology has lowered the cost of working on free software projects. As he puts it:

The problems with anarchism as a social system are also about transaction costs. But the digital revolution alters two aspects of political economy that have been otherwise invariant throughout human history. All software has zero marginal cost in the world of the Net, while the costs of social coordination have been so far reduced as to permit the rapid formation and dissolution of large-scale and highly diverse social groupings entirely without geographic limitation.¹³⁹

Professor Moglen therefore does not entirely reject economic analysis. He instead reminds us that to explain a given behavior one has to look at costs as well as incentives. To explain behavior that is essentially costless, or which costs very little, may not require much in the way of incentives. To say that the production of free software has increased as the cost of coordination and distribution has dropped is to say that members of the free software community are sensitive to the cost of programming relative to the benefits they receive from it. Even if many programmers are indifferent to reputational incentives and would write code solely because they enjoy it, to benefit other community members, or to dethrone a dominant firm such as Microsoft, it does not follow that

135. *Id.*

136. *Id.* § III.

137. *Id.* § III.

138. *Id.*

139. *Id.* § IV.

they would bear significant costs to do so. And even if open-source work did not cost programmers much, if the cost of coordination precluded production of a viable program the programmers might not bear even minimal cost without at least the satisfaction of seeing a useful result from their work.

Professor Moglen's emphasis on costs is important to making sense of the behavior we see. To refer to "free" or "open-source" software is to refer to a type of license. License terms affect the returns available for a given project, but they tell us nothing about the complexity of the code covered by the license, nor of the need or cost of coordinating work. Projects that employ open-source licenses might have little or nothing in common insofar as the cost of production is concerned. One would not expect the production of an operating system kernel consisting of a million lines of code to resemble the production of a network patch consisting of a relatively few lines of code, even if each project released its code under the same license. Moglen's analysis shows why it is important to evaluate the cost of particular projects rather than to speak generally of conditions in the production of "open-source" or "free" software.

Professor Moglen's references to the Linux kernel confirm this point. Although he insists that a search for conventional incentives misses the point, he does not dispute the forms of "ownership" Raymond identifies.¹⁴⁰ For example, Moglen describes the Linux kernel as an aggregation of developers "joined almost non-hierarchically in a development project ultimately involving more than one million lines of computer code—a scale of collaboration among geographically dispersed unpaid volunteers previously unimaginable in human history."¹⁴¹

Professor Moglen's qualification of his anarchic model—production on the Linux kernel is "almost" non-hierarchical—is important. To the extent Raymond correctly identifies norms that govern work on at least some projects, Moglen's anarchic explanation is incomplete. His cost-based analysis may explain why programmers will work on free software projects, but not why norms to coordinate work emerge and persist. The conditions that induce programmers to accept and work within a norm-based hierarchy, however, are a critical part of the answers to some of the questions we identified at the outset: whether the open-source/free software-production model is sustainable or generalizable.

We will return to this point shortly, but first it is useful to glance briefly at the Linux hierarchy. As we have seen, Red Hat told its investors that the Linux kernel was controlled by Linus Torvalds and a small group of engineers working with him.¹⁴² Torvalds has said that he was not motivated to start or maintain Linux by a desire for fame or reputational payoffs, though he acknowledges that such payoffs exist and some

140. See *id.* § III.

141. *Id.* § II.

142. See Red Hat S-1, *supra* note 36, at 6.

programmers value them.¹⁴³ Instead, Torvalds has said that he values being a member of a community working on interesting and socially desirable projects.¹⁴⁴ Consistent with Moglen's description, Torvalds surmised that many developers enjoy interacting with each other on the Internet and simply find programming enjoyable: "I really don't think you need all that much 'quid pro quo' in programming—most of the good programmers do programming not because they expect to get paid or get adulation by the public, but because it is *fun* to program."¹⁴⁵

Torvalds is equally clear, however, that there is a hierarchical aspect to work on the Linux kernel. As he says:

I don't think Linux is *inherently* better than FreeBSD or NetBSD. I just think that Linux is much more successful, partly because of better management, in my opinion. And because Linux has been more successful, there have been more people working on it, and it has developed a lot faster.

... [T]he fact that there is one person who everybody agrees is in charge (me) allows me to do more radical decisions than most other projects can allow.

For example, I can single-handedly decide that something is badly done, and re-do it completely even if it breaks lots of old code. It takes a while to recover from those kinds of decisions, but it makes for a better end result: if something is broken it gets fixed faster.¹⁴⁶

Linux is similar in this respect to work on the Perl scripting language, which one maintainer has described as a "constitutional monarchy," meaning that although the community of Perl developers debate the direction of the code and offer suggestions, the current maintainer decides to accept or reject them, subject to review and possible override by the project initiator.¹⁴⁷ On the other hand, decisions on Apache server software appear to be made by a committee of roughly two-dozen core developers, any one of whom may veto a decision.¹⁴⁸ Still, even a committee of two dozen is not radical decentralization. (This is particularly true if, as is probably the case, there are widely accepted criteria for evaluating many of the decisions the committee is asked to make; objective standards for decision would increase consensus and reduce the cost

143. See *FM Interview with Linus Torvalds: What Motivates Free Software Developers?*, 3 FIRST MONDAY (Mar. 1998), at http://www.firstmonday.org/issues3_3/torvalds/index.html (on file with the *University of Illinois Law Review*).

144. See *id.*

145. *Id.*

146. See Hiroo Yamagata, *The Pragmatist of Free Software: Linus Torvalds Interview*, at <http://www.tlug.gr.jp/docs/linus.html> (last updated Sept. 30, 1997) (on file with the *University of Illinois Law Review*).

147. Malcolm Maclachlan, *Panelists Describe Open Source Dictatorships*, INFO. WK. (Aug. 12, 1999), available at www.informationweek.com/story/TWB19990812S0003 (on file with the *University of Illinois Law Review*); see also Lerner & Tirole, *supra* note 65, at 12 (noting that Perl is administered on a rotating basis by ten to twenty programmers).

148. See Maclachlan, *supra* note 147; see also Lerner & Tirole, *supra* note 65, at 6 (noting the relatively centralized control over modifications to the "official" version of a project's code).

of group decision making.) As Lerner and Tirole conclude, “[i]n a number of instances, such as Linux, there is an undisputed leader. While certain aspects are delegated to others, a strong centralization of authority characterizes these projects.”¹⁴⁹

C. Academic Commentary

In addition to Professor Moglen’s analysis, the open-source community and its work have received a fair amount of academic attention. Professor Lessig sees open-source software as important to both our understanding of Internet governance and, possibly, to its future.¹⁵⁰ He uses open-source programming as an example of his broader theme that cyberspace is governed in important ways through its architecture, a key portion of which is software code.¹⁵¹ He describes open-source programming as a decentralized method of constructing the architecture of cyberspace, contrasted with the centralized model of closed development by single firms.¹⁵²

For Lessig, decentralized production implies that government will be less able to control open-source architecture than architecture produced by conventional firms.¹⁵³ He defines “closed code” as code controlled by “private for-profit organizations” and open code as code “outside of the control of any particular private for-profit corporation.”¹⁵⁴ He considers open code to be, in an informal sense, “commons code,” which is “neither privately-owned nor owned by the state, but is instead held in a commons,” the essence of which is that “no single person exercises an exclusive right over the code. Within the terms set by a range of licenses, anyone is free to take this code and develop it as he or she wishes.”¹⁵⁵ Lessig believes that governmental power to regulate open code is weak, while its power to regulate conventionally produced code is strong.

To the extent control over complex open-source projects is centralized, however, a government might be able to regulate an open-source project almost as easily as software produced by a conventional firm. If

149. Lerner & Tirole, *supra* note 65, at 22.

150. See Lawrence Lessig, *The Limits in Open Code: Regulatory Standards and the Future of the Net*, 14 BERKELEY TECH. L.J. 759, 764 (1999).

151. See *id.*

152. See *id.* at 764–65.

153. See *id.* at 764.

154. See *id.*

155. Lessig, *supra* note 150, at 764. Lessig is careful to note that his use of the “commons” terminology “is not technically accurate” though he believes “the spirit of the metaphor is correct.” As he says, and as we have seen, “to protect code from capture, software licenses place many conditions on the use of open code. Some conditions might seem technically inconsistent with the idea of a commons.” He therefore states that “perhaps a better description would involve a self-enforcing commons.” *Id.* As we have seen, authors own the rights in the code they write and are not themselves bound by “copyleft” provisions such as section 2(b) of the GNU GPL. See *supra* note 116. The informal commons to which Lessig refers would have to develop over time, as community members wrote new code and licensed it to each other.

the government wished to require that operating systems include certain code to facilitate government monitoring, for example, it is not immediately clear that a project maintainer such as Linus Torvalds would be a less effective focal point for prosecution than Microsoft. Of course, the principles on which at least a large portion of the open-source community operates are very different from the principles driving conventional software firms. While a conventional firm might silently acquiesce in a regulation that did not cost it too much, Torvalds and other maintainers of open-source projects might well rebel on grounds of principle.

The question then becomes how much risk project maintainers would bear. A priori analysis may not be much help on this point, though open-source norms do have a libertarian or anarchic tint to them. But if we assume that prosecutions or injunctions could effectively prevent maintainers from coordinating complex projects, we must then ask whether complex projects would continue to be developed and maintained through a truly decentralized production structure. The question is of course hypothetical, but the presence and persistence of central coordination of complex projects such as the Linux kernel gives us reason to be cautious on this point.

For smaller projects that are less complex and require less coordination, however, decentralized production might be quite robust and highly resistant to efforts to squelch code. The inability of courts to halt the distribution of hacker source code such as CPHack or DeCSS is evidence supporting this point.¹⁵⁶ The DeCSS case also suggests that open-source production might thwart government regulations even if the government succeeds in dictating the initial content of code. Government-mandated code might be disabled by small programs that require no coordination and thus would be extremely difficult to regulate. Lessig's decentralization point may therefore be correct for circumvention measures even if production of a large program could be regulated.

Other commentators have tended to describe open-source software as a reaction against property rights in code. Julie Cohen describes the open-source movement as a "consumer-driven development" caused by "a dissatisfaction with proprietary models for software development that emphasize intellectual property rights and discourage knowledge-sharing."¹⁵⁷ Peggy Radin and Polk Wagner describe the GNU GPL as a contract that runs with an object—the code—and thereby changes the social entitlement structure; Radin and Wagner describe the GPL as

156. E.g., Declan McCullagh, *Mattel Stays on the Offensive*, WIRED NEWS (Mar. 27, 2000), at <http://www.wired.com> (noting widespread mirroring of cphack, a hack of CyberPatrol, a program owned by a Mattel subsidiary) (on file with the *University of Illinois Law Review*).

157. Julie E. Cohen, Lochner in Cyberspace: *The New Economic Orthodoxy of "Rights Management"*, 97 MICH. L. REV. 462, 530 n.258 (1998). My reading of the evidence is that open-source production has thus far been driven by programmers and technically sophisticated users rather than consumers, though this of course may change if open-source production succeeds in creating consumer applications.

“cutting back” the scope of background intellectual property rights.¹⁵⁸ Others take a similar view, describing open-source software as “a clear reaction to the closed nature of copyright protected software,”¹⁵⁹ as a “unique licensing paradigm” that “works within the existing copyright doctrine to make [code] publicly available,”¹⁶⁰ or, relatedly, as the source of Microsoft’s worst fears.¹⁶¹ These discussions consider the relationship of open-source software, property rights, and competitive conditions, but do not focus on production as such.

As part of a larger effort to defend former article 2B of the Uniform Commercial Code—the proposed Uniform Computer Information Transactions Act (UCITA)¹⁶²—Robert Gomulkiewicz has correctly analyzed open-source production as a particular form of private ordering based on property rights: “The proponents of open source software rely on owning the copyright in the code and then licensing it according to a very particular mass-market licensing model.”¹⁶³ Gomulkiewicz thus rightly stresses that “[t]he distinction between open source software and typical commercial software is . . . based on the presence or absence of certain license terms.”¹⁶⁴

Because his larger project is to support the adoption of UCITA, Gomulkiewicz contends that the open-source software experience shows that mass-market licenses, and particular terms such as warranty disclaimers, must be enforced if software is to be efficiently developed and distributed to large numbers of users.¹⁶⁵ His article may be read as a response to scholars such as Pam Samuelson, who Gomulkiewicz says “equate the free flow of information with the placement of information in the public domain” and who “argue that licensing stifles information flow.”¹⁶⁶ He believes that “[t]he open-source software movement squarely refutes [this] premise” and that, “[a]rguably, licensing results in

158. Margaret Jane Radin & R. Polk Wagner, *The Myth of Private Ordering: Rediscovering Legal Realism in Cyberspace*, 73 CHI.-KENT L. REV. 1295, 1312–13 (1998). Radin and Wagner are quite right to focus on the social effects of the GNU GPL, as I discuss further in part IV.

159. Mark A. Haynes, Commentary, *Black Holes of Innovation in the Software Arts*, 14 BERKELEY TECH. L.J. 567, 573 (1999). Mr. Haynes, a practicing patent attorney, goes on to say that “the open source movement exploits and is hindered by copyright.” *Id.*

160. Ira V. Heffan, Note, *Copyleft: Licensing Collaborative Works in the Digital Age*, 49 STAN. L. REV. 1487, 1491 (1997).

161. See Stewart Minor Benjamin, *Stepping Into the Same River Twice: Rapidly Changing Facts and the Appellate Process*, 78 TEX. L. REV. 269, 299–300 (1999).

162. UNIF. COMPUTER INFO. TRANSACTIONS ACT (UCITA) §§ 101–904 (West 2000).

163. Robert W. Gomulkiewicz, *How Copyleft Uses License Rights to Succeed in the Open Source Software Revolution and the Implications for Article 2B*, 36 HOUS. L. REV. 179, 185–86 (1999).

164. *Id.* at 186.

165. *Id.* at 194. Gomulkiewicz states:

The open source software revolution uses licensing to perpetuate what it considers a superior software development model and to provide low cost software to the mass market. Whether open source software will become the “next great thing” that endures, only time will tell. Licensing will be at the center of its success or failure.

Id.

166. *Id.* at 182 n.15.

greater information flow than would be the case if publishers were forced to give up their information for free.”¹⁶⁷

Gomulkiewicz makes many good points in his article, but this conclusion overstates his case. Open-source licenses indeed rest on property rights, but they do not actively exploit the core right to exclude. At least the GPL vests that right in an author who holds it in reserve as a method of enforcing adherence to the norms embodied in the license. Conventional licenses actively employ the right to exclude by trading a partial sacrifice of that right—permission to use code—for money. There is an important difference between active exclusion and the threat of exclusion designed to enforce a regime of “free” copying, modification, and distribution. Thus, whereas both open-source production and conventional production do rest on property rights, one cannot infer from the GPL that conventional licenses provide greater public access to information and innovations than a regime of free copying. Perhaps this is the case, but establishing such a claim requires independent arguments. The GPL does not prove such a claim. To the extent the GPL increases the flow of information relative to public-domain software, it is because the GPL denies licensees the active use of the right to exclude that typifies conventional licenses.¹⁶⁸

Professors DeLong and Froomkin discuss open-source production in the context of a broader analysis of what they see as causes or potential causes of failure in markets for digital technologies.¹⁶⁹ They identify a lack of transparency—“imperfect information”—as a potential cause of failure “because much of the value added in the data-processing and data-communication industries today comes from complicated and evolving systems of information provision.”¹⁷⁰ From this perspective, “open source solves the software opacity problem with total transparency; the source code itself is available to all for examination and re-use.”¹⁷¹ DeLong and Froomkin see variation among open-source licenses as “a source of potential opacity” but say that “it is easy to envision a world in which open-source software plays an increasingly large role.”¹⁷²

As part of their explanation of how traditional law and economics principles would assess open-source programming, DeLong and Froomkin focus on the relationship between property rights, production structure, and agency costs, confirming that a neoclassical explanation of

167. *Id.*

168. As Gomulkiewicz recognizes elsewhere in his article, the GPL does not place information in the public domain. *See id.* at 186–87.

169. J. Bradford DeLong & A. Michael Froomkin, *Speculative Microeconomics for Tomorrow's Economy*, 5 FIRST MONDAY (Feb. 2000), at http://www.firstmonday.org/issues/issue5_2/delong/index.html (on file with the *University of Illinois Law Review*).

170. *Id.*

171. *Id.*

172. *Id.*

open-source production would focus on reputational payoffs to programmers:

Open source's vulnerability (and perhaps also its strength) is that it is, by design, only minimally excludable. Without excludability, it is harder to get paid for your work. Traditional economic thinking suggests that all other things being equal, people will tend to concentrate their efforts on things that get them paid

Essentially volunteer software development would seem particularly vulnerable to the tragedy of the commons. Open source has, however, evolved a number of strategies that at least ameliorate, and may even overcome, this problem. Open-source authors gain status by writing code. Not only do they receive kudos, but the work can be used as a way to gain marketable reputation. Writing open-source code that becomes widely used and accepted serves as a virtual business card, and helps overcome the lack of transparency in the market for high-level software engineers.¹⁷³

DeLong and Froomkin are right to say that conventional economic analysis explains open-source production by linking programmer incentives as closely as possible with labor market returns. On this account, open-source programmers are really investing in their future income, implying that they will do so only so long as the present value of that future income exceeds their current cost. Raymond also notes this possibility, saying that "occasionally, the reputation one gains in the hacker culture can spill over into the real world in economically significant ways. It can get you a better job offer, or a consulting contract, or a book deal."¹⁷⁴ He contends, however, that such payoffs are "rare and at best marginal for most hackers; far too much so to make it convincing as a sole explanation, even if we ignore the repeated protestations by hackers that they're doing what they do not for money but out of idealism or love."¹⁷⁵ On this point, Raymond, Torvalds, and Moglen concur.¹⁷⁶

IV. IMPLICATIONS

This part examines some of the implications of open-source licenses and norms for the theory of the firm and for the law.

173. *Id.* DeLong and Froomkin go on to note that compensation in reputation may not be enough to support development in large open-source projects and thereby might reduce the number of features available to users, though this in turn might not be much of a problem because most users use only a few features of most programs. *Id.*

174. Raymond, *Homesteading the Noosphere*, *supra* note 104, § 5.

175. *Id.*

176. In addition, Lerner and Tirole have recently argued that many aspects of open-source production can be explained through the economic literature on "career concerns," which posits a "career concern incentive" for which payoffs are job offers and, perhaps, shares in open-source firms. Lerner and Tirole distinguish this incentive from the "ego gratification incentive," for which the payoff is peer recognition. They refer to both incentives as the "signaling incentive," and to this extent their model is similar to Raymond's and to DeLong and Froomkin's explanation of how conventional economic models would explain open-source production. Lerner & Tirole, *supra* note 65, at 3.

A. Firms

From an economic perspective, the open-source software community and firms such as Red Hat may seem trivial and uninteresting. If developers are simply investing in reputation with an eye toward increasing future wages,¹⁷⁷ then we do not even need to say that they are maximizing anything other than income. If Raymond is right to say that this motivates only a small fraction of community members, then perhaps we need only say that they presumably value reputation for its own sake, and therefore maximize reputation instead of income.¹⁷⁸ Because reputation may be traded for income, however, it is impossible to say that programmers in general either do or do not value it for its own sake. Probably programmers both enjoy reputation for its own sake and like the idea that they can cash in on it if they wish; these are not “either-or” incentives. In any event, both approaches fit easily within the rational-actor assumption and standard neoclassical tenets.¹⁷⁹

Even this explanation, however, produces some interesting questions and predictions. Suppose open-source programmers wish to maximize reputation, either for its own sake or for income. What sort of reputation do programmers wish to acquire, and how do they acquire it? If the reputation desired is one for programming expertise, then programmers should seek to work on the most technically difficult problems, which might or might not be the ones of most pressing concern to social welfare. If the reputation is one for management expertise as well as programming, then programmers should seek opportunities to run their own open-source projects as maintainers. If the reputation is one for teamwork and cooperation, then perhaps any program large enough to require cooperation would do.

Maximizing reputation could lead to a suboptimal distribution of programming resources within the community. Difficult technical problems whose solutions yield little practical utility are an obvious example. Working on such problems might enhance a programmer’s reputation for expertise but might do little for social welfare (at least in the short run). On the other hand, suppose that for each potential aspect of reputation one could expect programmers to gravitate to projects worked on by the largest number of other programmers. This would give each programmer the largest audience for their work, and therefore the strongest repu-

177. *Id.* at 15 (naming this incentive “the career concern incentive”).

178. *See id.* (naming this “the ego gratification incentive”).

179. Or as Raymond puts it:

The Linux world behaves in many respects like a free market or an ecology, a collection of selfish agents attempting to maximize utility which in the process produces a self-correcting spontaneous order more elaborate and efficient than any amount of central planning could have achieved. . . . The “utility function” Linux hackers are maximizing is not classically economic, but is the intangible of their own ego satisfaction and reputation among other hackers.

Eric S. Raymond, *The Cathedral and the Bazaar*, § 10, at <http://www.tuxedo.org> (restructured Aug. 2000) (on file with the *University of Illinois Law Review*).

tational payoff.¹⁸⁰ If we assume that programs worked on by large numbers of programmers are likely to be socially useful, the desire for widespread recognition might offset any maldistribution of programmer resources. Solving difficult problems on large projects is likely to produce broad benefits. And if ordinary hackers may move among projects at low cost, or work on several simultaneously, lock-in or path-dependence effects seem unlikely.¹⁸¹

But suppose this supposition is wrong. Reputation only comes from being the programmer to solve a problem. Large numbers of programmers working on a project make up an audience, but they are also competitors. Suppose the probability that a given programmer's solution to a problem is chosen for incorporation into a project is inversely related to the number of other programmers offering solutions to the same problem. Then the expected value of work on large projects might be low, even if the potential payoff from being the programmer with the best solution would be high. Risk-averse programmers therefore might spend too much time searching for small projects they could dominate, for which the reputational payoffs might have a higher expected value even if the nominal payoff was lower.

We have little actual evidence on these points. A recent survey of open-source development found that most developers work on only one or a few projects, concluding that “[f]ree software development is less a bazaar of several developers involved in several projects” and “more a collation of projects developed single-mindedly by a large number of authors.”¹⁸² Although this conclusion suggests that developers concentrate their efforts, it does not shed much light on the variables that lead developers to choose their projects. This survey found that the three authors that have contributed the most open-source code are institutions—the FSF, Sun Microsystems (Sun), and the University of California (UC)—suggesting that institutional mandates have to be taken into account when assessing developer incentives.¹⁸³

Staying with the reputational payoff hypothesis for a moment, we should examine how the norms of the open-source community might provide reputational payoffs to programmers who demand them. The provisions of the GNU GPL that require credit to be given to programmers, for example, secure reputational payoffs, as does the norm against

180. Lerner and Tirole make this point as well. See Lerner & Tirole, *supra* note 65, at 15.

181. This analysis is analogous to the work of Axelrod, Mitchell, Thomas, Bennett, and Bruder, modeling the incentives for firms to join standards-setting alliances. See Robert Axelrod et al., *Coalition Formation in Standard-Setting Alliances*, in ROBERT AXELROD, *THE COMPLEXITY OF COOPERATION* 96 (1997). This piece presents a brief case study of UNIX fragmentation that is useful in understanding the open-source community.

182. Rishab Gosh & Vipul Ved Prakash, *The Orbitten Free Software Survey*, 5 *FIRST MONDAY* (July 2000), at http://www.firstmonday.org/issues/issue5_7/ghoshold/index.html (on file with the *University of Illinois Law Review*).

183. See *id.*

removing a programmer's name from a project.¹⁸⁴ More interestingly, Raymond suggests that Torvald's success with the Linux kernel is based in part on his frequent revisions and distribution of the code.¹⁸⁵ In the kernel's early days, "it wasn't unknown for him to release a new kernel more than once a *day*."¹⁸⁶ However one specifies reputational incentives, this practice reduced the time between the programmer's effort and her payoff: Programmers who simply enjoyed programming would see their work being put to use immediately and thereby gain immediate satisfaction from a job well done, and those seeking to enhance reputation would become well-known more quickly. As elsewhere, shortening the wait for a payoff increases its value.¹⁸⁷

But we must not accept the reputation explanation of open-source production too quickly. In the first place, because so much open-source work is done by programmers at institutions such as the FSF, Sun, and UC, it is difficult to separate programmers' individual motivations from their institutional motivations. This might not be much of a problem if we ask only why programmers work on open-source projects, because any reputational gains to an employee from such work increase the value of the employee's outside option, and therefore fit comfortably within standard economic explanations. But institutional motivations make it hard if not impossible to infer individual incentives from the type of projects to which programmers devote their time.

More seriously, some aspects of the reputation-based model of programming generate predictions about programmer behavior that contradict the behavior community members describe. The contradiction suggests that, even for complex projects requiring coordination, reputational payoffs are only part of the open-source story. Assume that programmers desire to develop reputations for programming expertise, managerial expertise, or both. Assume further that project maintainers (such as Linus Torvalds) receive greater reputational returns than ordinary programmers. On these assumptions, rational programmers maximizing reputational gains would compete for the maintainer's slot on high-profile projects.

184. Cutting against these points is the finding of the Orbitten survey that "credits are often not available" and "rarely follow a set format." *Id.*

185. See Raymond, *The Cathedral and the Bazaar*, *supra* note 179, § 4.

186. *Id.*

187. One might also see this practice as lowering the risk of opportunism by shortening the duration of the transaction—the relevant unit of analysis from a transaction cost perspective—and by the quicker accumulation of information about the good faith of the project manager and therefore about the security of the reputational return. This practice also presents a problem: Many if not most commercial users have no interest in daily revisions of their operating system. They may be interested only in function, not code, and want revisions only when the code has been sufficiently improved to improve function in a way meaningful to the firm's business. This likely state of affairs implies demand for an intermediary between the open-source community and the commercial world. The intermediary could monitor developments of the code, aggregate and double-check it, and release new versions only when improvements in function warranted it. That is one niche Red Hat seeks to fill.

Because open-source licenses allow forking, these assumptions imply that such programmers will engage in strategic forking to enhance their reputational returns. If programmers care only about the size of the project they work on—and thus the extent of their reputational gain—one would expect programmers to accept strategic forking and work on the project run by the maintainer who most credibly promised to maximize programmer reputations on her project. But I have seen no reports of cases in which forking occurred simply to enhance reputation. To the contrary, Raymond suggests that forking is quite rare: “There is strong social pressure against forking projects. It does not happen except under plea of dire necessity, with much public self-justification, and with a renaming.”¹⁸⁸

One could counter this by speculating that programmers seek to maximize a reputation for cooperativeness, but unless that trait is scarcer in programmers than in society in general, one would expect a reputation for team play to add relatively little if any value to a programmer’s outside option. A more likely speculation is that the norm developed because the open-source community wishes to distinguish itself from the competitive model of conventional software production. Norms against forking reward conduct that makes production of code smoother and collectively more efficient and penalize conduct for which individual returns are likely (on average) to exceed the gains to the code base. Such norms tend to keep the community’s focus on code rather than individual income.

The hierarchical structure of some open-source projects is also in tension with the notion that programmers maximize only reputation. As shown by Torvalds’ comments on the benefits of his sole control of Linux, whether open-source programmers receive reputational returns is, to a degree, up to the project maintainer. Even if the maintainer is an honest broker, interested only in the technical quality of code, a programmer’s work could be disregarded at the maintainer’s option, presumably lowering the programmer’s reputational gains. Perhaps programmers accept the hierarchy in return for the broader audience that pays attention to large projects run by maintainers such as Torvalds, but it may also be that programmers derive value other than reputation from their work on such projects.¹⁸⁹

188. Raymond, *Homesteading the Noosphere*, *supra* note 104, § 3.

189. Loss of returns due to project hierarchies may not be as great a problem as it seems at first. The Orbiten survey finds that the base of active programmers is relatively narrow, with 10% of total authors (1,271) writing 72% of the total open-source code. The survey credited the top ten authors (.08%) with 19.8% of the code. As noted above, the largest authors in this survey are institutions, and we do not know how many individual programmers contributed to this work. Even so, however, the point in the text has to be qualified, perhaps significantly, by the fact that individual authors might earn status within their institution for doing work even if a project maintainer ultimately rejected it.

These percentages might also reflect project maintainers economizing on information costs by preferring submissions from proven programmers over submissions from newcomers. The need for coordination implies some form of hierarchy because information is costly to process and evaluate. *E.g.*,

Pointing out that the reputational payoff model is incomplete is useful, but it moves us no closer to answering our important questions. Although reputational payoffs imply more strategic behavior than we see, to characterize individual programmers as simply maximizing the joy of programming does not explain the centralized production structures we see in complex projects. Some portions of the open-source community are made up of persons and tasks that sort themselves out in a manner that looks a lot like firms. Why do these structures emerge, are they sustainable, and is the model generalizable?

Let us first consider whether open-source hierarchies may in part be sustained by a form of asset specificity. Assume that some open-source projects provide rewards (of whatever type) that programmers cannot get by working on other projects. This might be the case, for example, for projects of extraordinary importance to the community, or which directly challenge a dominant firm, or which (perhaps for these reasons) attract the attention of large numbers of hackers. The Linux kernel would satisfy each of these conditions.

For such cases, and as a heuristic, let us examine an analogy based on Hart and Moore's contention that employees are more amenable to fiat control than are independent contractors. They argue that the employment relation involves the owner of an asset allowing the agent to use assets the agent does not have but which are necessary for the agent to maximize his or her productivity.¹⁹⁰ Hart and Moore contend that employees will be induced to act at least partially for the benefit of their principal if the principal controls access to such an asset: Employees who are at least partially faithful "put themselves in a stronger bargaining position later on with the" principal "who determines whether they have access to the asset."¹⁹¹

Hart and Moore's argument explains why employees would be more likely than independent contractors to accept fiat control: Even though both could, in theory, terminate their relations at will, the employee's returns depend on her relationship with the principal to a greater extent than do the independent contractor's returns. To persuade the principal to continue granting the agent access to the assets,

KENNETH J. ARROW, *THE LIMITS OF ORGANIZATION* 39–42, 53–55 (1974). It is cheaper for Linus Torvalds to evaluate forty submitted solutions for a problem than to have every programmer working on the Linux kernel evaluate them. The Linux hierarchy lowers the information cost of producing the kernel, which, in Coasean terms, is why the hierarchy exists. Presumably the Linux hierarchy—with Torvalds having the final say and the labor of maintaining different parts of the kernel to a degree divided among lieutenants—is designed to review code in the way that produces efficient decisions, by which we mean decisions based on information acquired and reviewed only to the point where the marginal benefit exceeds the marginal cost. If programmers understand that hierarchies are necessary for large projects to exist at all, then they might accept the risk of having their submissions rejected as inseparable from the gains they seek.

190. In the independent contractor relationship, by contrast, each side owns the assets they need to be productive.

191. Oliver Hart & John Moore, *Property Rights and the Nature of the Firm*, 98 J. POL. ECON. 1119, 1121 (1990).

the agent is, to a degree, willing to work diligently on the principal's behalf. We may extend the point in a way that makes the analogy to open-source production more plausible: Employees whose skills are more valuable when deployed with a particular array of assets (a particular firm) should be able to bargain for relatively higher wages at the firm and therefore should be willing to accept some exercise of fiat power within the firm. This argument implies generally that an employee's loyalty will vary with the degree to which her productivity depends on firm-specific assets, which at least roughly corresponds to what we see in employment markets.

For open-source software production, this analysis implies that programmers will be more likely to accept hierarchy in projects that provide unique or unusually valuable returns. A programmer who had a strong desire to temper Microsoft's power, for example, would be more likely to accept the hierarchy of the Linux kernel than a programmer who cared nothing about Microsoft. The challenge the GNU/Linux OS presents to Microsoft would be an asset providing a productive yield unavailable from other projects. Programmers motivated by reputational concerns might be willing to tolerate hierarchy in very large projects even if they would be unwilling to accept it in smaller projects.

We must be careful, however, not to overstate this point. I refer to it as a heuristic because it helps us distinguish among types of projects to which a programmer might devote his time, and to distinguish among types of programmers who might be drawn to particular projects. But open-source programmers generally are not employed by the projects they work on. In Hart and Moore's model, employees look after the employer's interests (if only partially) in order to ensure future access to the assets the employees need to maximize productivity. There is no analogue to this point in open-source production. The terms of open-source licenses effectively preclude project maintainers from denying code to programmers, and programmers therefore cannot be induced to accept hierarchy by the threat of being cut off from the relevant code. Indeed, it is hard to think of any threat more likely to destroy a project. The point here is only that projects that offer unique returns will be better able to sustain hierarchies than those that do not.

With reputation and asset-specificity being only partial answers, and with the latter not being able to explain why preferences form in the first place, we must look deeper into the community structure. Herbert Simon has written that the question for firms "is not whether free riders exist—much less employees who exert something less than their maximum—but why there is anything *besides* free riding."¹⁹² He posits loyalty—the identification of workers with the goals of the firm—as an important answer to this question. Arguing that "human beings are not the

192. Simon, *supra* note 21, at 34.

independent windowless Leibnizian monads sometimes conjured up by libertarian theory.”¹⁹³ Simon suggests:

[Organizational loyalty and pride rest] upon a discrimination between a “we” and a “they.” Identification with the “we,” which may be a family, a company, a city, a nation, or the local baseball team, allows individuals to experience satisfaction (to gain utility) from successes of the unit thus selected. Thus, organizational identification becomes a motivation for employees to work actively for organizational goals.¹⁹⁴

For Simon, this point is importantly related to authority within organizations, for authority is often used “to coordinate behavior by promulgating standards and rules of the road, thus allowing actors to form more stable expectations about the behavior of the environment (including the behavior of other actors).”¹⁹⁵ In other words, while formal economic analysis in general and game theory in particular are extremely helpful in analyzing agent interactions, we also need to understand how preferences are formed and constrained by the institutional setting in which actors act.¹⁹⁶

The lower cost of communication among members of the open-source community should increase the amount of communication and enhance each member’s sense of belonging to a community. Torvalds mentions this point as one of the chief benefits of work on open-source projects, and it underlies much of Professor Moglen’s analysis. But the open-source community is defined both by what it is and what it is not. Open-source work is opposed to conventional “closed” development, and many members of the community couch this opposition in very strong terms. As we have seen, Richard Stallman believes the difference between free software and conventional “proprietary” software is a fundamental matter of social ethics: He says “the aim of the GNU project is to create a better community.”¹⁹⁷ Not surprisingly, many of the FSF’s arguments focus on the ways in which free software is more socially desirable than conventional production, using conventional production as a

193. *Id.* at 35.

194. *Id.* at 36.

195. *Id.* at 39.

196. As Douglass North says:

[G]ame theory highlights the problems of cooperation and explores specific strategies that alter the payoffs to the players. But there is a vast gap between the relatively clean, precise, and simple world of game theory and the complex, imprecise, and fumbling way by which human beings have gone about structuring human interaction. Moreover, game theoretic models, like neoclassical models, assume wealth-maximizing players. But as some of the experimental economics literature demonstrates, human behavior is clearly more complicated than can be encompassed in such a simple behavioral assumption.

DOUGLASS C. NORTH, INSTITUTIONS, INSTITUTIONAL CHANGE AND ECONOMIC PERFORMANCE 15 (1991).

197. Hiroo Yamagata, *Better Society Through Free Software: Richard M. Stallman Interview*, at <http://tlug.linux.or.jp/rms.html> (last updated Sept. 30, 1997) (on file with the *University of Illinois Law Review*).

foil and a method of distinguishing the community of free software programmers.

Even for community members who do not agree with the ethical claims of the FSF, it is easier to see the advantages of open-source programming, and to develop a strong sense of community among open-source programmers, when they have the conventional model of binaries-only distribution resting upon active use of the right to exclude to use as a contrast. Thus one programmer commented on Raymond's *Homesteading the Noosphere* by positing that:

[Open-source licenses are] a defensive line drawn between our culture and the exchange culture. Basically what they say is that the legal system of the surrounding culture is not going to be permitted to intrude into ours and that we are willing to use artifacts of the exchange culture (namely their notion of intellectual property) to enforce this.¹⁹⁸

For this hacker, the sense of an open-source community formed in opposition to the "other" of conventional programming was quite strong:

Hacker culture has a lot of power, but it's small and based on an at least somewhat vulnerable foundation (post scarcity economics). It also doesn't have its own system of force, whereas the surrounding exchange culture is fairly well-armed with a legal system and legal force to back up the dictates of that system. As such, we're in constant danger of being swept away by the ocean.¹⁹⁹

To the extent the open-source community coheres by discriminating between itself and the "other" of conventional programming, Microsoft may play an important role in supporting the development of production hierarchies on complex projects such as the GNU/Linux OS. Raymond, for example, distances himself from the FSF's ethical condemnation of conventional programming models while positing that "pragmatic" hackers discriminate to some degree among conventional firms:

The typical pragmatist attitude is only moderately anticommercial, and its major grievance against the corporate world is not 'hoarding' per se. . . . If the pragmatist hates anything, it is less likely to be 'hoarders' in general than the current King Log of the software establishment, formerly IBM, now Microsoft.²⁰⁰

Several of the most successful open-source projects are plausibly opposed to Microsoft's market position. The basic architecture of the Internet is largely open-source, and the Internet itself has rightly been seen as a serious threat to Microsoft.²⁰¹ More directly, the GNU/Linux

198. Russ Allbery, *Comments on Homesteading The Noosphere*, at <http://www.eyrie.org/~eagle/writing/homesteading.html> (last modified July 5, 1998) (on file with the *University of Illinois Law Review*).

199. *Id.*

200. Raymond, *Homesteading the Noosphere*, *supra* note 104, § 2.

201. *See id.* This threat, however, is diminishing as Microsoft adapts its business to the Internet-based marketplace. For anecdotal evidence that Mr. Torvalds and the GNU/Linux OS community are

OS is a potential substitute for Microsoft Windows and NT. Programmers working on these projects might see themselves as part of a community resisting Microsoft's conventional business model. Having such an aggressive and dominant opponent probably contributes to the sense of community among hackers and may well provide a convenient focal point—the “other”—that the open-source and free software communities may use to define themselves.

To the extent Simon's conception of discrimination and loyalty are important to sustaining the hierarchies that coordinate more complex open-source projects, and to the extent that Microsoft serves as an “other,” we must ask whether open-source production of complex projects is viable where those projects do not compete with Microsoft. For example, can open-source production of complex applications succeed? Though Microsoft dominates several applications markets, these are less important to its market position than its operating system, and there are many applications niches in which Microsoft is not a dominant firm. Some evidence suggests that open-source methods can succeed even without the benefit of Microsoft as a direct foil. One might mention some of the fundamental architecture of the Internet and the Web, such as TCP/IP, HTML, Apache, and Sendmail. Still, for very complex projects such as the GNU shell or the Linux kernel, the question is an open one that should be watched to see what we may learn as production moves farther away from technologies that square off against the current dominant firm.

The matter of applications raises two questions: Does the open-source model work only for projects that present technical challenges that interest hackers (or, relatedly, does it work only for sophisticated technology of greater interest to hackers than to general consumers)?; and will open-source production work for “polishing” projects to make them easier for unsophisticated users to use, or will programmers stop supporting projects once the interesting technical problems have been solved? Opinions on these questions vary. We do not have very good evidence on them at present, and, therefore, they are best treated as sources of some predictions to guide further observation.

A reputation-based model predicts that uninteresting questions will not attract programmer interest because reputations are not made by tedious work on average problems. The notion that hackers hack for fun suggests a similar dropping-off of support, at least if one assumes that it is less fun to work on user interfaces than on core technical issues. More community-based theories suggest that programmers will continue to

motivated by opposition to Microsoft, see Lee Gomes, *Torvalds Plans Major Upgrade of Linux Operating System*, WALL ST. J., Aug. 18, 2000, available at <http://www.wsj.com/archive/retrieve.cgi?id=SB966551209980435711.djm> (on file with the *University of Illinois Law Review*) (noting that Microsoft tests showing that Windows NT outperformed Linux on certain measures had been “very motivational” to the Linux community and had prompted fixes to code bottlenecks that had caused Linux to lose the tests).

work on even relatively uninteresting projects that are important to the community; this is particularly true of projects that might compete with Microsoft.

Then there is the question of getting a project started. In his first paper, Raymond noted that Linux was based on UNIX designs. He said it would be very hard to write an open-source project from the ground up because a “nascent developer community needs something . . . to play with.”²⁰² This raises the question whether a traditional commercial model is needed to underwrite enough code to jump-start large open-source projects. UNIX was of course originally funded by Bell Laboratories at AT&T, and much of the important work building on AT&T’s start was done by programmers at Berkeley under a license from AT&T that included source code.²⁰³ Berkeley’s work is probably best understood as a straight subsidy, as is government funding of important aspects of the Internet, which creates the cost structure that makes the open-source community viable. The viability of large, unsubsidized open-source projects is an open question.

Commercial open-source firms also present some risks to this social structure. Nothing we have seen alters the need for programmers to obtain some return for their cost of work. Many programmers work full-time for firms, doing open-source work on their own time or with the blessing of their firms.²⁰⁴ The Orbiten survey’s conclusion that Sun is the second leading author of open-source code—behind the FSF, an organization dedicated to the production of such code—tends to confirm this point.²⁰⁵ Academics might enjoy similar support, as the University of California’s status as the third leading author of code suggests. So many, and probably most, programmers do not face penury because of their open-source work. But because cost is the value of the option foregone, commercial open-source firms could increase the stress on this structure by raising the returns to work on closed projects that complement, for example, the GNU/Linux OS.²⁰⁶

Suppose, for example, that Red Hat hires dozens of highly skilled open-source coders. Its business model requires it to make money through service, or by tailoring GNU/Linux systems to fit the needs of a particular client. If its employees are to maximize the value of their

202. Raymond, *The Cathedral and the Bazaar*, *supra* note 179, § 9.

203. For a brief history, including a description of the important license between AT&T and Berkeley, see *Unix Systems Laboratories, Inc. v. Berkeley Software Design, Inc.*, 832 F. Supp. 790, 793–94 (D.N.J. 1993).

204. Many firms, especially those that deal with or compete against Microsoft, might give their blessing quite readily.

205. See Gosh & Prakash, *supra* note 182.

206. This may be one reason for some criticism of Red Hat and VA Linux within the open-source community, though there are plenty of other potential explanations as well. See Red Hat S-1, *supra* note 36, at 7 (“Some members of the open source software community have criticized the expansion of our strategic focus These critics argue that our strategy could fragment the Linux community into a variety of competing factions, resulting in a less cohesive and cooperative development process.”).

shares, they must devote to paid work time they might otherwise have spent coding on open-source projects. If we assume a fairly close relationship between such efforts and employee income, then there is a risk that commercial open-source firms will siphon resources away from open-source projects.

Commercial open-source firms thus might move the community closer to the conventional economic model described by DeLong and Froomkin, in which community members emphasize the realizable monetary value of their reputations with an eager eye to the paying commercial world, similar to some law students or college athletes. On the other hand, firms that make their living selling services complementary to open-source projects certainly have a stake in the health of those projects, and it would make a lot of sense for such firms to allow their agents to spend time working on open-source projects. The status (reputation) of such firms in the open-source community is important to their business models in a variety of ways, and both Red Hat and VA Linux say that their employees devote substantial time to open-source work. Still, this analysis does identify a problem requiring management within the community that bears watching.²⁰⁷

After identifying all these points on which conclusions should await more evidence, let us return to some points about which we may be reasonably confident. Most clearly, the Internet made the open-source production model viable by lowering the costs of coordination.²⁰⁸ These costs include identifying a problem to be tackled, recruiting programmers, receiving and incorporating their input, and giving reputational or other forms of social returns to programmers who require them. Open-source production may therefore be seen as an example in which high coordination costs led to the creation of firms devoted to conventional production models — the expected Coasean result — but in which a reduction in those costs allowed the formation of less centralized, unconventional “firms” devoted to particular projects. Though hierarchy persists

207. Red Hat's efforts to give programmers who had contributed to the Linux kernel a chance to buy into the Red Hat IPO provide a good example of the cultural tension between the open source community and financial markets. See C. Scott Ananian, *A Linux Lament* (July 30, 1999), at http://www.salon.com/tech/feature/1999/07/30/redhat_shares/index.html (on file with the *University of Illinois Law Review*); C. Scott Ananian, *Inside the Red Hat IPO* (Aug. 13, 1999), at http://www.salon.com/tech/feature/1999/08/13/redhat_shares/index.html (on file with the *University of Illinois Law Review*).

208. As Red Hat puts it:
[I]n the software industry, the internet is profoundly changing the way that software is developed and distributed. The internet has enabled multiple groups of developers to collaborate on specific projects from remote locations around the globe. Developers can write code alone or in groups, make their code available over the internet, give and receive comments on other developers' code and modify it accordingly. The internet has also provided an avenue not only for less expensive and speedier delivery of code, but also for support and other online services. . . . The growth of the internet has greatly increased the scale and efficiency of open source development through the availability of collaborative technologies such as e-mail lists, news groups and web sites. These technologies have enabled increasingly large communities of independent developers to collaborate on more complex open source projects.

Red Hat S-1, *supra* note 36, at 39; see also Moglen, *supra* note 122.

in these firms, the hierarchies appear to be very narrow bands sitting atop a relatively decentralized set of producing agents, a similarly expected Coasean result.

By making widespread distribution of code possible, technology also allows programmers who maximize reputation to reap greater returns from their work. Though programmers on these projects are not subject to fiat control by project maintainers,²⁰⁹ projects may command a degree of loyalty by providing a forum for programmers to develop their reputational capital or to satisfy an urge to create in a social context supporting that creation. The social structure of the projects themselves is therefore a large part of the point of open-source production. To some and perhaps many programmers, the community of a given project—including its hierarchy—may be as relevant a product of their work as the code itself. For open-source production, therefore, the social consequences of cost-reducing technology are as significant as the ability of such technology to lower the cost of production.

With respect to small projects that do not require extensive coordination, logic and experience to date suggest that the open-source model is sustainable. For interesting problems encountered by a large number of programmers—such as fixes to popular hardware or software—the model is likely to be quite robust. We cannot in the abstract say how sustainable or robust the model is for complex projects that require coordination and, therefore, some form of hierarchy. Complex projects are costlier to produce than simple ones and therefore require stronger supporting structures if production is to go forward. We may say that projects that are important to the community as a whole are more likely to succeed than niche projects; projects that compete with a dominant and aggressive firm are more likely to succeed than those that do not; projects that present interesting technical questions are more likely to succeed than projects that do not; and projects that attract large numbers of developers and provide them with reliable feedback and reputational payoffs are more likely to succeed than projects that do not. In each case, these variables relate to the conditions necessary for the formation of norms that support hierarchical production models outside the traditional firm context.

We may also say that the payoffs these variables provide will have to track, in at least a rough way, the complexity and cost of the project. Greater project complexity implies a more complex hierarchy, which is costlier to administer and carries a greater risk of error. These factors may reduce some payoffs to programmers. The success of such projects will, in any event, depend on whether their maintainers can minimize coordination costs—importantly including the cost of processing and evaluating information—relative to conventional production. To the de-

209. Programmers at institutions, however, may be subject to such control by their institution.

gree that software production becomes componentized, for example, coordination costs will decrease and open-source production of components will become more robust. Coordination and assembly of components will still be costly, of course. But Red Hat's viability to date is some evidence that even now firms can provide such services without the active use of exclusion to facilitate conventional software production.

B. *Intellectual Property Rights*

Open-source software production depends importantly on property rights. The GNU GPL system rests on the assignment of property rights (copyright) in an author, who is then able to grant nonexclusive, transferable rights to community members, subject to limitations that enforce community tenets. This structure gives subsequent users of the copyrighted code the ability to pass along restrictions that embody open-source tenets, resulting in the dissemination of the tenets in proportion to the distribution of the code. The right to exclude is not abandoned, however: This model gives the rights-holder the ability to enforce open-source tenets through an infringement action if necessary. Professor Moglen is quite right to give Richard Stallman and the GNU project significant credit for this elegant use of the existing property rights structure. It is, however, a structure that ultimately rests on a property right.

Indeed, if we assume that there will always be opportunistic programmers who might try to appropriate a base of open-source code for use in a conventional program,²¹⁰ then open-source production will always have to rely on the right to exclude being vested in a person or entity willing to wield that right to enforce community norms and thwart appropriation of the community's work. Otherwise programmers could not be confident that outsiders—members of the conventional software world—would not free ride on the programmers' work. Without such confidence, programmers might be reluctant to devote their time to open-source programming. While programmers may do quite a lot of work simply for the joy of it, their views might change if someone else were making money off of their labor.

The open-source model, however, may conflict with copyright doctrine in some respects. Cases have recognized a fair use defense to infringement where code was copied by transformative users who wished to make their works compatible with existing works;²¹¹ even the Digital Millennium Copyright Act includes a narrowed version of the traditional defense.²¹² This authority extends (though just barely) to copying to cre-

210. Or, relatedly, programmers nearing the end of their careers who care little for their reputation in the developer community, an instance of the familiar "end-period" problem.

211. See *Sony Computer Entm't, Inc. v. Connexix Corp.*, 203 F.3d 596, 603–10 (9th Cir. 2000); *Sega Enters. v. Accolade, Inc.*, 977 F.2d 1510, 1520–28 (9th Cir. 1992).

212. See 17 U.S.C. § 1201(f) (1994); *Universal Studios, Inc. v. Reimerdes*, 111 F. Supp. 2d 346, 346–47 (S.D.N.Y. 2000) (issuing preliminary injunction and rejecting § 1201(f) defense to claims based

ate a familiar user interface as well.²¹³ Depending on the facts of particular cases, the GNU GPL's requirement that works incorporating GPL-ed code also be subject to the GPL might run afoul of this authority.

Speculating on fair use in hypothetical cases is not very interesting, because such determinations must rest on a careful examination of actual facts in particular cases. My point is only that the open-source model of production is itself a fact that should be considered. The cases that have been decided have featured conflicts between firms working within the conventional software production model. In context, the copying in those cases was done to expand the range of choices that the particular model provided to consumers. A private firm that copied open-source code to work with a conventional, binaries-only application might also expand choices available to consumers. But in that case the copying would bridge production models as well as programs.

In one respect, the crossing of models might not matter. One may always argue that a copier such as Borland or Accolade is to some extent free-riding on the work of someone else. That a private firm might take advantage of the work of open-source programmers is therefore not a unique problem. But broader questions remain. Because copyright policy seeks ultimately to provide creative works to the public,²¹⁴ copying of freely distributable source code to enhance the production of for-profit binary distributions deserves careful consideration. A standard fair use inquiry would ask in part how the copying would affect the market for the copied work. This question is designed to ascertain whether the copying would reduce the author's returns by a notable amount. The question makes sense where the copied work was produced by an author—for software, generally a firm—operating under the conventional model of production. Where code may be freely copied and distributed, however, asking how copying affects its market is unlikely to result in useful analysis. With no traditional “firm” involved in the production of open-source code, one would hope that a court would also ask how a particular instance of copying would affect the social structure underlying open-source production of the copied code.²¹⁵

on distribution of DeCSS, a program alleged to circumvent the Content Scramble System (CSS) for DVDs).

213. See *Lotus Dev. Corp. v. Borland Int'l, Inc.*, 49 F.3d 807, 821 (1st Cir. 1995), *aff'd*, 516 U.S. 233 (1996).

214. See authorities cited in *supra* note 30.

215. Patent law obviously has a number of implications for this model, but examining them would require a fair-sized article in its own right. I therefore do not discuss patent law here, other than to note that the FSF and the GPL consider software patents a threat to their work and that the preamble to the GPL states “that any patent must be licensed for everyone's free use or not licensed at all.” *GNU General Public License*, *supra* note 48, preamble. For an assessment of the current state of software patents, see Julie Cohen & Mark A. Lemley, *Patent Scope and Innovation in the Software Industry*, 89 CAL. L. REV. (forthcoming 2001), available at http://www.papers.ssrn.com/paper.taf?ABSTRACT_ID=209668 (on file with the *University of Illinois Law Review*).

C. *Contract Law*

The key to open-source production is the way that property rights are deployed by the various licenses. The legal status of those licenses is therefore an important question, which I explore here using the GPL as an example. The GPL model seeks to create binding obligations on downstream code through notice-and-use provisions. Its terms are designed to follow the code from user to user. This is not an expansion of property rights, however. The code is copyrighted and the GPL grants limited, nonexclusive permission to use it. If the GPL is ineffective, the copyright still persists.

The first doctrinal issue is whether the GPL's notice-plus-conduct model is sufficient to form a contract. Recall that the GPL requires licensees to include the GPL terms with code they distribute; this tells downstream users that they may not copy or modify the code unless they accept the terms. The GPL states that by modifying or copying GPL-ed code, a user agrees to be bound by the GPL.²¹⁶

From one perspective, the question whether this model works is only marginally interesting. The code is subject to copyright; the effect of the GPL is to give downstream users a defense to an infringement action by the rights holder. Setting aside the fair use questions discussed in the last section, a downstream user who claimed the GPL did not bind her would merely open herself to an infringement action. Downstream users would therefore *want* the GPL to be enforceable.

As a doctrinal matter, one may wonder whether downstream users can be bound by the GPL even though they are not in privity with the rights holder,²¹⁷ or whether the GPL model provides notice sufficient to form an agreement even if privity is not a barrier. But as a practical matter, we are unlikely to see downstream users claiming the GPL is unenforceable unless the downstream user is claiming a fair use right to copy code in a manner at odds with the GPL. The question is more likely to come up, if it comes up at all, in a case of opportunistic behavior by a licensor. The question would then be whether the rights granted by the GPL may be terminated at the author's will.

With respect to formation questions, the closest cases to the GPL involve shrinkwrap licenses. Courts and commentators seem to be in something of a muddle over shrinkwraps. In part, the muddle results from a failure to distinguish different contract-related issues that have

216. Recall the GPL's statement to downstream users:

You are not required to accept this license, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

GNU General Public License, *supra* note 48, ¶ 5.

217. *Merges*, *supra* note 34, at 118–21.

come up in cases involving shrinkwraps. A case that finds against a licensor on the ground that a contract was formed before shrinkwrap terms were made available²¹⁸ may be cited together with a case holding that a particular term is preempted by federal law (in which no formation issue is discussed)²¹⁹ and a case finding that a shrinkwrap formed a contract but that it was one of sale and not a license.²²⁰ Each case may find against the rights-holder, but that fact does not justify categorical statements about shrinkwraps.

Though the literature is muddled, some basic principles seem clear. The notice-plus-conduct model may *form* a contract between parties who both know or reasonably should know that some of the terms of their agreement will be provided only after the initial order and payment. Whether particular terms may be enforced in particular cases is, as it should be, a separate question. As to formation alone, however, the rule that a contract may be formed “in any manner sufficient to show agreement, including conduct by parties which recognizes the existence of such a contract” is a familiar one that may be readily applied in this context.²²¹ The key is notice. If *A* understands (or reasonably should understand) that *B* is willing to deal only on terms that will not be provided until after some initial terms have been agreed to and performance has begun then, assuming a right to return, the (lawful) additional terms are part of the parties’ agreement unless *A* rejects them when he sees them. In that case, there is no agreement and the parties must take steps to place each other in the positions they were in before performance began. If *A* does not wish to begin performance before seeing all the terms, he may demand to see them first.

So long as this structure is clear to both parties from the outset, the notice-plus-conduct model operates in much the way as the theory of formation for printed forms. Parties who receive printed forms and do not read them understand that they are agreeing to the substance of the transaction—parking or renting a car, flying on a plane, or obtaining a

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

219. *Vault Corp. v. Quaid Software, Ltd.*, 847 F.2d 255, 269–70 (5th Cir. 1988).

220. *Novell, Inc. v. Network Trade Ctr., Inc.*, 25 F. Supp. 2d 1218, 1230 (D. Utah 1997), *vacated in part*, 187 F.R.D. 657 (D. Utah 1999).

221. U.C.C. § 2-204(1) (2000). UCITA adopts this language and adds the “layered contracting” concept found in its section 208(2), which states that: “[T]he terms of a record may be adopted . . . after beginning performance or use *if the parties had reason to know that their agreement would be represented in whole or in part by a later record . . .*” UNIF. COMPUTER INFO. TRANSACTIONS ACT (UCITA) § 208(2) (West 2000) (emphasis added). Comment 3 to section 208 makes clear that this “layered contracting” approach gives effect to terms received after performance begins only “if at the time of initial agreement, the parties had reason to know and, thus, expected that this would occur and that terms . . . to be agreed would provide elaboration of their contract.” *Id.* § 208 cmt 3. By contrast, “[I]f, instead, the parties consider the deal to be closed at the outset, subsequently proposed terms from either party are treated as proposed modification of the agreement, effective only under concepts applicable to such modifications.” *Id.* At the same time, however, UCITA declines to adopt one of the important rules justifying legal recognition of assent to form agreements, which makes its overall approach to formation questionable. *See infra* note 223.

credit card—plus some contract terms of unknown content. They know that they do not know all the relevant terms, and they agree to proceed on that basis. Professor Llewellyn reached the heart of the matter:

Instead of thinking about ‘assent’ to boiler-plate clauses, we can recognize that so far as concerns the specific, there is no assent at all. What has in fact been assented to specifically are the few dickered terms, the broad type of the transaction, and but one thing more. The one thing more is a blanket assent (not specific assent) to any not unreasonable or indecent terms the seller may have on his form, which do not alter or eviscerate the reasonable meaning of the dickered terms.²²²

In Professor Llewellyn’s model, parties in such circumstances are protected from opportunism and abuse by doctrines such as unconscionability and unfair surprise.²²³

This summary fairly captures the reasoning and results of most cases on the subject. Consider *Step-Saver Data Systems, Inc. v. Wyse Technology*,²²⁴ which is sometimes cited as rejecting formation based on this model. The plaintiff was a value-added reseller that ordered software from the defendant. The software malfunctioned, and the plaintiff sued

222. KARL N. LLEWELLYN, *THE COMMON LAW TRADITION: DECIDING APPEALS* 370 (1960). One must of course also note Professor Llewellyn’s description of the problem a few pages earlier. Referring to the form agreement, he says:

It would be a heart-warming scene, a triumph of private attention to what is essentially private self-government in the lesser transactions of life or in those areas too specialized for the blunt, slow tools of the legislature—if only all businessmen and their lawyers would be reasonable.

But power, like greed, if it does not always corrupt, goes easily to the head. So that the form-agreements tend either at once or over the years, and often by whole lines of trade, into a massive and almost terrifying jug-handled character; the one party lays his head into the mouth of a lion—either, and mostly, without reading the fine print, or occasionally in hope and expectation (not infrequently solid) that it will be a sweet and gentle lion.

Id. at 362; see also E. ALLAN FARNSWORTH, *CONTRACTS* § 4.26, at 301–02 (3d ed. 1999).

223. See *RESTATEMENT (SECOND) OF CONTRACTS* § 211(3) (1979). UCITA refuses to adopt the doctrine of unfair surprise, as embodied in *RESTATEMENT* § 211(3). This omission skews the balance drawn in Professor Llewellyn’s measured defense of assent in a form-contract environment. If one is to be bound to terms that a drafter fully expects will not be read, then legal recognition of assent should be limited to terms reasonably related to the economic substance of the transaction, for these are the only sort of terms to which assent may reasonably be inferred from the substance of the transaction. For a defense of UCITA’s omission of this provision, see generally Holly K. Towle, *The Politics of Licensing Law*, 36 *HOUS. L. REV.* 121 (1999) (basing her critique largely on Professor White’s analysis of how courts have used section 211(3), which itself focuses largely on Arizona, which produced over half the cases interpreting the provision). See also James J. White, *Form Contracts Under Revised Article 2*, 75 *WASH. U. L.Q.* 315 (1997) (explaining his opposition to the proposed addition of the reasonable expectation principles of section 211(3) to article 2 of the UCC). Professor White does say that if Arizona’s experience “foretell[s] the future of form contracts,” then “sellers, lessors, lenders and the many others whose agreements fall under Revised 2-206 and 2A-206 have justified trepidation about the Uniform Commercial Code’s recognition of the doctrine of ‘reasonable expectations’ in those provisions.” *Id.* at 326. But Professor White attributes Arizona’s experience largely to an activist judge on the state Supreme Court and to the Arizona courts’ failure to apply the principles embodied in section 211(3) rigorously. See *id.* at 346–47. The combination of activism and inattention has created a body of precedent repeating the misapplication. See *id.* Misapplication of legal rules by activist judges is always a risk, however, and Professor White’s condemnation of the rule is therefore not as severe as it might at first appear.

224. 939 F.2d 91 (3d Cir. 1991).

for breach of warranty. The defendant's shrinkwraps disclaimed all warranties; the question was whether the disclaimer was part of the parties' agreement.²²⁵

The decision in *Step-Saver* rests mainly on the court's analysis of when the parties' agreement was formed. Step-Saver ordered software over the telephone from the defendant.

[The defendant, TSL, would] accept the order and promise, while on the telephone, to ship the goods promptly. After the telephone order, Step-Saver would send a purchase order, detailing the items to be purchased, their price, and shipping and payment terms. TSL would ship the order promptly, along with an invoice. The invoice would contain terms essentially identical with those on Step-Saver's purchase order: price, quantity, and shipping and payment terms. No reference was made during the telephone calls, or on either the purchase orders or the invoices with respect to a disclaimer of warranties.²²⁶

Step-Saver argued that the parties' agreement was formed during its telephone conversations with TSL. It argued that TSL's shrinkwraps materially altered the terms of the telephone agreement and therefore, under section 2-207 of the U.C.C., were only a proposal for additional terms and not a part of the agreement.²²⁷ Though the court did not specify exactly when it thought a contract had been formed, it did find that formation preceded delivery of the shrinkwrap terms. Finding that "Step-Saver never expressly agreed to the terms of the box-top license, either as a final expression of, or a modification to, the parties' agreement," the court held that the disclaimers were not part of the parties' agreement.²²⁸

Step-Saver was complicated by evidence that the defendant had twice tried to get the plaintiff to sign agreements containing warranty disclaimers, and that the plaintiff had refused both times,²²⁹ that the defendant continued to deliver software to the plaintiff even after these refusals, that the defendant's agent had told Step-Saver that the shrinkwrap terms applied only to end-users and not Step-Saver, and that both parties agreed that some of the form terms written on the packaging did not apply to their particular transaction.²³⁰ Perhaps because of these facts, the *Step-Saver* court carefully qualified the extent of its holding:

TSL has raised a number of public policy arguments focusing on the effect on the software industry of an adverse holding concerning the enforceability of the box-top license. We are not persuaded that requiring software companies to stand behind their products will inevitably destroy the software industry. We emphasize, however,

225. *See id.* at 97.

226. *Id.* at 96.

227. *See id.* at 97.

228. *Id.* at 98.

229. *See id.* at 102-03.

230. *See id.* at 103.

that we are following the well-established distinction between conspicuous disclaimers made available before the contract is formed and disclaimers made available only after the contract is formed.²³¹

This quotation does not say what the court would have done if the parties' initial discussions had included a caution that additional terms would be forthcoming, but had not included the full text of the disclaimers. The court was certainly not sympathetic to shrinkwrap terms in general, and to warranty disclaimers in particular, so perhaps such additional facts would not have mattered. But *Step-Saver* does not preclude the formation of a contract in which the parties agree that they will begin performance subject to approval of terms to be reviewed later.²³² Indeed, because the court's analysis rejects such terms only "to the extent that the terms of the writing either add to, or differ from, the term detailed in the parties earlier writings or discussions,"²³³ the case is at least open to a the notice-plus-conduct model where proper notice of additional terms is given at the outset.

Cases approving of the notice-plus-conduct model tend to support enforceability of the GNU GPL, though these cases are arguably too willing to find adequate notice that a licensor would provide additional terms to a licensee after performance had begun. In *ProCD v. Zeidenberg*,²³⁴ for example, the Seventh Circuit found a contract where terms on the outside of a software package gave purchasers notice that further terms were contained inside the package. The court stated that if the terms on the outside of the box were part of the parties' agreement (as Zeidenberg agreed) then the agreement included the notice that additional terms were contained inside the box.²³⁵ From a formation perspective, this is best seen as an instance of enforcing a form agreement referring to additional terms. Legal recognition of assent in such a case may be justified on the theory Professor Llewellyn articulated.

231. *Id.* at 104.

232. *Step-Saver* is often cited in conjunction with *Arizona Retail Sys., Inc. v. The Software Link, Inc.*, 831 F. Supp. 759 (D. Ariz. 1993). The plaintiff there was a reseller that configured systems; the defendant sold a multi-user operating system. The plaintiff ordered an evaluation copy of the defendant's code, and then ordered several copies for use. Though the case is most often cited as rejecting the shrinkwrap model, the court actually found that license terms printed on the outside of the envelope containing the evaluation software were in fact part of the parties' agreement. (The court was no doubt influenced in this conclusion by evidence that an agent of the plaintiff actually read the terms and concluded that they were not enforceable.) The court reached the contrary conclusion on the subsequent orders however. These orders typically involved the plaintiff telephoning the defendant and requesting shipment of a copy, which the defendant shipped in its shrinkwrap. Following *Step-Saver*, the court concluded that on these occasions contracts were formed by the order and shipment, and that the shrinkwraps were proposals for additional terms. If the defendant had told the plaintiff on these occasions that the deal would be subject to additional terms, the court's result might well have been different. Because the court found that an agent of the plaintiff had read the terms when it received the evaluation software, it is curious that the court did not consider whether the plaintiff knew or should have known that subsequent orders would be subject to similar terms.

233. *Step-Saver Data Sys.*, 939 F.2d at 99.

234. 86 F.3d 1447, 1450-51 (7th Cir. 1996) (Easterbrook, J.)

235. *Id.*

*Hill v. Gateway 2000*²³⁶ extended this analysis—indeed, perhaps overextended it—to a case in which the licensee did not have a package warning that additional terms would follow after payment and delivery of a computer.²³⁷ *Hill* involved a mail-order purchase of retail hardware and software by an end-user. The purchase was arranged over the telephone; the plaintiffs specified the desired product and provided the defendant with a credit card number, and the defendant agreed to ship the product.²³⁸ The plaintiffs in *Hill* apparently did not brief the question whether the lack of any notice of additional terms distinguished the case from *ProCD*, though the point did come up in oral argument. Judge Easterbrook disposed of it quickly, stating first that the difference between the box-top notice in *ProCD* and the absence of notice on the box in *Hill* was “functional, not legal.”²³⁹ By this he meant that *ProCD* involved a purchase in a retail store, and the box in that case was designed for that purpose. The box in *Hill*, by contrast, was a shipping carton that bore only such legends as “Fragile” and “This side up.”²⁴⁰ This distinction is no doubt true, but it is hard to see what it has to do with the information the parties had when they agreed to begin performance. The plaintiffs’ point was about ex ante information, not boxes.

236. 105 F.3d 1147 (7th Cir. 1997).

237. In *Hill*, Judge Easterbrook said that *ProCD* had already decided the matter. *Id.* at 1149. *ProCD*, however, refers to terms on the outside of the software package giving notice that further terms would be found inside. *See ProCD*, 86 F.3d at 1451. Thus the statement in that case that “notice on the outside, terms on the inside, and a right to return the software for a refund if the terms are unacceptable . . . may be a means of doing business valuable to buyers and sellers alike.” *Id.*

238. *Hill*, 105 F.3d at 1147.

A customer picks up the phone, orders a computer, and gives a credit card number. Presently a box arrives, containing the computer and a list of terms, said to govern unless the customer returns the computer within 30 days. Are these terms effective as the parties’ contract, or is the contract term-free because the order-taker did not read any terms over the phone and elicit the customer’s assent?

Id. Judge Easterbrook concluded that:

Practical considerations support allowing vendors to enclose the full legal terms with their products. Cashiers cannot be expected to read legal documents to customers before ringing up sales. If the staff at the other end of the phone for direct-sales operations such as Gateway’s had to read the four-page statement of terms before taking the buyer’s credit card number, the droning voice would anesthetize rather than enlighten many potential buyers. Others would hang up in a rage over the waste of their time. And oral recitation would not avoid customers’ assertions (whether true or feigned) that the clerk did not read term X to them, or that they did not remember or understand it. Writing provides benefits for both sides of commercial transactions. Customers as a group are better off when vendors skip costly and ineffectual steps such as telephonic recitation, and use instead a simple approve-or-return device.

Id. at 1149. All this is quite true, and a good reason for not requiring full disclosure of all terms before payment. It does not explain, however, why Gateway could not have mentioned to customers before charging their credit cards that Gateway was willing to deal only on additional terms that would be shipped with the computer. Although such notice might have made no difference to either the plaintiff in *Hill* or to most other customers, there is no basis in the theory or doctrine of contract law for enforcing terms that follow performance unless the facts show that the parties knew or reasonably should have known that such terms would be coming.

239. *Id.* at 1150.

240. *Id.*

The court in any event concluded that the plaintiffs did have notice, and that *ProCD* therefore applied.²⁴¹ Judge Easterbrook wrote that “the Hills knew before they ordered the computer that the carton would include some important terms, and they did not seek to discover these in advance.”²⁴² The court based this factual conclusion on the ground that “Gateway’s ads state that their products come with limited warranties and lifetime support.”²⁴³ Because the advertisements did not specify the content of these terms, which the court reasonably concluded would be more important to most consumers than the arbitration clause at issue in the case, the court found that the ads sufficiently notified consumers that additional terms were in the offing. The Hills chose not to ask for these terms in advance but to review them when the computer arrived. Thus, “by keeping the computer beyond 30 days” (the time specified in the shrinkwrap), “the Hills accepted Gateway’s offer, including the arbitration clause” at issue in the case.²⁴⁴

Perhaps Judge Easterbrook thought that the use of shrinkwrap terms in retail computer transactions was so well-established that the Hills either knew or should have known that additional terms would be coming. As he put it, “payment preceding the revelation of full terms is common for air transportation, insurance, and many other endeavors.”²⁴⁵ More formally, perhaps Judge Easterbrook believed the Hills should have known additional terms would be coming because that is part of the usage of trade for retail computer sales.²⁴⁶ The opinion referred to no evidence of such usage, however, and the U.C.C. requires that the existence and scope of usage be proved as facts.²⁴⁷ Absent some factual basis for concluding that the Hills knew or should have known when they placed their telephone order that Gateway would only deal on additional terms to be provided with the computer, *Hill* goes too far. Nevertheless, in *Brower v. Gateway 2000, Inc.*,²⁴⁸ a New York court followed *Hill* on the question of formation. Properly distinguishing between this question and the enforceability of particular terms, however, the *Brower* court held unconscionable a term specifying a particularly expensive arbitrator (whose fees exceeded the cost of most of Gateway’s products) that followed England’s “loser pays” rule and that required that all correspondence be sent to its headquarters in France.²⁴⁹

241. *Id.*

242. *Id.*

243. *Id.*

244. *Id.*

245. *Id.* at 1149.

246. This was the analysis of the Washington Supreme Court in the *Mortenson* case, which we examine in a moment.

247. U.C.C. § 1-205(2) (2000). The question whether the law should recognize as usages of trade practices that rights holders in general find advantageous, but to which end-users object, is an interesting one, but beyond the scope of this article.

248. 676 N.Y.S.2d 569 (N.Y. App. Div. 1998).

249. *Id.* at 574.

Most recently, in *M.A. Mortenson Co. v. Timberline Software Corp.*,²⁵⁰ the Washington Supreme Court relied on a course of dealing and usage of trade analysis to satisfy the notice requirement and enforce shrinkwrap terms. Much as in *Step-Saver*, the plaintiff had issued a purchase order (which did not contain an integration clause) and received software. The code came in plastic pouches with Timberline's license terms printed on the outside. Notice of the terms was also displayed on the first screen when the software was run.²⁵¹ One of these terms was a disclaimer of liability.²⁵² The software contained a bug that caused one of Mortenson's bids for a construction project to be \$1.9 million too low.²⁵³ Mortenson sued, and the question was whether Timberline's disclaimer was part of the parties' agreement.²⁵⁴ The court held that the parties' conduct as a whole had formed an agreement, even though the precise moment of formation could not be specified.²⁵⁵

Much can be said about the analysis in *Mortenson*; I cannot address it in full here. For our purposes, the important thing is that the court's legal analysis rested in significant part on a factual question: Did Mortenson know or have reason to know when it sent its purchase order that the software it received would be subject to additional terms?²⁵⁶ Based on the parties' previous course of dealing (the software in question was an upgrade to software Mortenson had previously obtained from Timberline), and what it decided was a usage of trade, the court held that Mortenson did.²⁵⁷

Whether the GNU GPL forms an enforceable license in a particular case therefore depends on whether the rights-holder and intermediate licensors give adequate notice of the GPL terms to downstream users when code is transferred.²⁵⁸ The GPL requires each user to "conspicu-

250. 998 P.2d 305 (Wash. 2000).

251. *Id.* at 308.

252. *Id.* at 308-09.

253. *Id.* at 309.

254. *Id.* at 310-11.

255. *Id.* at 313.

256. *Id.*

257. Mortenson did not introduce evidence contradicting Timberline's assertion of a usage of trade, allowing the Court to conclude that there was no genuine issue of material fact on the point and, therefore, to affirm the trial court's decision granting summary judgment to Timberline. *See id.* at 314. The district court in *Klocek v. Gateway, Inc.* declined to follow *ProCD*, *Hill*, and *Mortenson*, on the ground that U.C.C. § 2-207 applied to a consumer's telephonic purchase of a computer. *Klocek v. Gateway, Inc.*, 104 F. Supp. 2d 1332, 1337 (D. Kan. 2000). The court found that an agreement had been formed on the telephone, and the terms shipped with the software amounted to a proposal for additional terms, which the consumer did not accept. *Id.* at 1339-42.

258. Which makes troubling the observation that:

Big hunks of code, often with hacks in place, are sent back and forth and all around, all the time, without the GPL being attached. No doubt the parentage of this code is sometimes in doubt. It would not be at all surprising to learn that some of it had found its way into commercial software, distributed without source code, in violation of the GPL.

Dennis E. Powell, *comment: Judgment Day for the GPL? Determining the Legality of the GPL*, LINUX PLANET, at <http://www.linuxplanet.com/linuxplanet/reports/2000/1/> (last visited Aug. 28, 2000) (on file with the *University of Illinois Law Review*).

ously and appropriately publish on each copy” of code “an appropriate copyright notice and disclaimer of warranty.”²⁵⁹ Persons distributing modified code must comply with this term and, if the code works interactively, cause the modified code to display a copyright notice when it is run and tell users how to view a copy of the GPL.²⁶⁰

If the terms of the GPL are placed on distributed code, the formation question resembles a standard form contract situation. A copyright notice combined with directions to a website or other source from which the GPL may be viewed extends the standard form contract model, but it could still form a binding agreement so long as the notices were adequate to let a reasonable recipient know that the person distributing the code intended the GPL to bind downstream users.²⁶¹ A usage of trade analysis might play a role here as well; particularly in cases where code was transferred among hackers, a reference to the GPL combined with a link to a webpage posting its full terms might be sufficiently well understood to justify an inference of assent, even if the full terms of the GPL were not included on the code.

Even if a court found no basis for inferring an enforceable agreement, however, hackers who devoted time and effort to a project that was billed as subject to the GPL might be able to assert equitable claims to prevent rights-holders from misusing the hackers’ work. For example, suppose that an author distributed code under the GPL and members of the open-source community worked to improve the code by fixing flaws and writing additional code. Suppose further that, at some point after considerable improvements have been made, the author claims that the GPL is ineffective to grant enforceable licenses to create derivative works. If the author attempted to take the improved version of the code private, equitable theories such as estoppel might provide a useful back-stop in cases where the facts could not support a formal contract theory.²⁶²

Three other contract-related issues are worth mentioning. The first is privity of contract. Professor Merges has suggested that the GPL may not bind downstream users who take code from someone other than the

259. *GNU General Public License*, *supra* note 48, § 1.

260. *Id.* § 2(c).

261. UCITA provides that, in Internet-type transactions, users will be considered to have had an opportunity to review standard-form terms if the terms are made available for review before information is provided to the licensee, or before the licensee becomes obligated to pay, whichever comes first. UNIF. COMPUTER INFO. TRANSACTIONS ACT (UCITA) § 211(1) (West 2000). Terms are deemed to have been made available if the terms “or a reference to an electronic location from which they can readily be obtained” are displayed in close proximity to the information to be licensed. *Id.* § 212(1)(a). (By “reference” the statute presumably has in mind a hyperlink.) The GPL may or may not fit this model precisely, depending on the circumstances in which particular code is made available, but the procedure it sets forth seems within the spirit of this provision.

262. RESTATEMENT (SECOND) OF CONTRACTS § 90; FARNSWORTH, *supra* note 222, § 2.19. Such theories would still require notice, of course, for a programmer who had no reason to believe a project was subject to the GPL could not have relied on the GPL’s terms in deciding to devote time to the project.

rights-holder because persons who encounter license terms that run with the code will not be in privity with the rights-holder.²⁶³ Technology might reduce this problem to some degree. If a rights-holder posted open-source code and the GPL terms on a website and each community member downloaded the code from that site, each licensee would be in privity with the rights-holder. Centralized distribution of code is at odds with the freedom of hackers to share code directly with each other, however, and the marginal gain in enforceability might not be worth the sacrifice in community practices.

The privity concern might not be dispositive in any event. The GPL is a nonexclusive, transferable license. (Or, if one prefers, a license granting authority to sublicense.) One might view distribution of GPL code as simply a transfer within the terms of a license. An analogy might be drawn to a licensee who holds the rights to distribute a work in North America, who may contract with regional firms, and who may contract with individual venues for the performance of the work.²⁶⁴ In addition, one may view authors of derivative works as the licensors of at least their improvements to a program and perhaps of the derivative work as a whole (though this latter point is less clear). A court holding this view of matters would probably not view a lack of privity as a barrier to finding an enforceable agreement.

Then there is the question of the license term. The GNU GPL states no term for the rights it grants and limitations it imposes. Two circuits have held that a license that states no term is terminable according to applicable state law. In each case that meant the license was terminable at the licensor's will.²⁶⁵ The Ninth Circuit concluded that a license that states no term has an implied term of thirty-five years. The court based this odd holding on a provision in the Copyright Act that gives authors a five-year window (beginning in the thirty-fifth year) to terminate a license agreement regardless of the license terms or state contract law.²⁶⁶ The court's statutory interpretation was poor, and both the Seventh and Eleventh Circuits have rightly declined to follow the Ninth on this point. In light of this authority, outside the Ninth Circuit rights-

263. *Merges*, *supra* note 34, at 128–29.

264. There will always be at least one party in privity with the rights-holder, of course—the initial transferee of the code. One might therefore be tempted to dismiss the privity concern on the ground that all that is needed is for the original transferee to bring suit. But if we posit an opportunistic rights-holder, we must consider that the decision to breach the GPL and take code private might only set up a bargaining game. In other words, if only the initial transferee could keep the rights-holder from privatizing code, then the rights-holder need only buy off the initial transferee. This is true of all rights-holders of course, but the bargaining problems multiply rapidly with the number of licensees who hold potentially valid claims. If one wishes to prevent opportunism, therefore, one would prefer that a large number of rights-holders have valid claims.

265. See *Korman v. HBC Fla., Inc.*, 182 F.3d 1291, 1295 (11th Cir. 1999); *Walthal v. Rusk*, 172 F.2d 481, 485 (7th Cir. 1999).

266. See *Rano v. Sipa Express, Inc.*, 987 F.2d 580, 585 (9th Cir. 1993); 17 U.S.C. § 203(a)(3) (1994).

holders may terminate the rights of GPL licensees pursuant to applicable state law, which may mean termination at will in many cases.

At least in theory, the potential ability to terminate at will increases the risk of opportunistic behavior by rights holders. The problem could be easily remedied by adding a term specifying the duration of the GPL rights. Such a term could have an information-forcing effect: Rights-holders who wished to preserve the option of terminating the GPL would have to use an earlier version of the license, signaling to community members that they faced a greater risk of opportunistic behavior from that project than from projects employing the modified license.²⁶⁷ The possibility that GPL rights are terminable at will suggests one odd result—the hacker community might be better off if the licenses were unenforceable as formal contracts, for then community members could rest on estoppel arguments. If a contract is formed, however, at least outside the Ninth Circuit the community would be subject to the applicable state default rules and the possibility that GPL rights could be terminated.

Termination may or may not present a very great practical risk, depending on the code in question. An initial author could terminate the GPL rights she had granted to use and modify her code, but not the rights licensees had in the code they wrote to form a derivative work. This much seems clear from *Stewart v. Abend*,²⁶⁸ which states that “the aspects of a derivative work added by the derivative author are that author’s property, but the element drawn from the pre-existing work remains on grant from the owner of the pre-existing work,”²⁶⁹ and from § 103(b) of the Copyright Act, as codified, which provides that “[t]he copyright in a . . . derivative work . . . does not imply any exclusive right in the preexisting material . . . and does not affect or enlarge the scope, duration, ownership, or subsistence of, any copyright protection in the preexisting material.”²⁷⁰

So, to return to our earlier example, if Regan may terminate the GPL rights she grants, then she may prevent Cordelia from distributing Regan’s original code in Cordelia’s derivative work. Presumably, termination would be effective as against persons receiving Cordelia’s work under the GPL, for Cordelia could not give them greater rights with re-

267. For more background on this, see generally Ian Ayres & Robert Gertner, *Filling Gaps in Incomplete Contracts: An Economic Theory of Default Rules*, 99 YALE L.J. 87, 91–94, 101–04 (1989); Ian Ayres & Robert Gertner, *Strategic Contractual Inefficiency and the Optimal Choice of Legal Rules*, 101 YALE L.J. 729 (1992); Jason S. Johnston, *Strategic Bargaining and the Economic Theory of Contract Default Rules*, 100 YALE L.J. 615, 624 (1990).

268. 495 U.S. 207, 223 (1990).

269. That the original code might have been intermingled with the licensee’s code does not vitiate the termination right (if one exists under state law):

So long as the pre-existing work remains out of the public domain, its use is infringing if one who employs the work does not have a valid license or assignment for use of the pre-existing work. It is irrelevant whether the pre-existing work is inseparably intertwined with the derivative work.

Id. (citations omitted).

270. 17 U.S.C. § 103(b) (1994).

spect to Regan's code than Cordelia had herself. Cordelia could, however, continue to distribute her own code, as to which she held the rights. Whether Regan's termination was a large blow to the project as a whole would depend on how important her original code was. Whether Cordelia's code would be worth anything without Regan's would depend on the same thing. Whether Regan would be likely to terminate would depend at least in part on whether she needed to use Cordelia's code. A termination by Regan could invite reciprocal termination by Cordelia. Projects that incorporate code from many authors, therefore, seem unlikely candidates for unilateral termination.²⁷¹

Lastly, there is the question of assignments. One way to combat opportunism is to ask authors of open-source code to assign their rights to an organization controlled by a representative portion of the community. The organization would then decide whether to terminate rights or take code private and, if properly constituted, its decisions might reasonably reflect community sentiment. This is indeed the model that the Free Software Foundation advocates.²⁷² Assignments might present some risk to authors, however, as shown by a recent case involving a hack of "CyberPatrol," an Internet filter marketed by Microsystems, Inc.²⁷³ Two hackers, Eddy Jansson and Matthew Skala, decided to disassemble CyberPatrol to see how it worked and what it blocked. The two wrote three programs, two based on Linux and one on Windows, that revealed the list of websites that CyberPatrol blocked as well as other information about CyberPatrol's operation. The programs could be used to ascertain a password that in turn could be used to disable CyberPatrol.²⁷⁴ Messrs. Jansson and Skala released the programs, as well as a memorandum explaining their work, in a posting entitled "cp4break."²⁷⁵ Jansson wrote the Windows-based program, entitled CPHack. One source file in that program stated: "CPHack v0.1.0 by Eddy L O Jansson

271. To add a touch of legal realism, one suspects that few judges would be very eager to wield their injunctive powers to terminate one of such an interconnected web of rights.

272. See Declan McCullagh, *Mattel Ruling Confuses Hackers*, WIRED NEWS (Mar. 29, 2000), at <http://www.wired.com> (quoting Professor Moglen) (on file with the *University of Illinois Law Review*). Authors may pursue this model now, of course, and the decision to assign copyrights to a body accepted by the community might serve a signaling function equivalent in some respects to modifying the GPL to specify a term. I am not aware of data on what percentage of open-source authors have assigned their rights, however, and therefore cannot assess whether this option serves as a meaningful signal within the community. And the political differences within the open-source community appear to some degree to involve the FSF, which might cause some authors who would accept the assignment model in general to decline to assign rights to the FSF in particular. This fact makes it difficult to use the FSF to assess whether assignment is useful as a signal to combat opportunism.

273. *Id.*

274. Typically, a parent would choose a password that could be employed to disable the program when adults wished to view blocked sites.

275. Matthew Skala, *Cyber Patrol Break FAQ*, at <http://www.islandnet.com/~mskala/cpbfaq.html> (last visited Aug. 28, 2000) (on file with the *University of Illinois Law Review*).

/ Released under the GPL.”²⁷⁶ Jansson added this message on his own; he meant to tell Skala he had done so, but he forgot.²⁷⁷

Microsystems responded to the distribution of these programs with a suit against the hackers for copyright infringement, breach of contract, and interference with prospective economic advantage.²⁷⁸ A trial court in Boston issued a temporary restraining order against the hackers.²⁷⁹ Jansson and Skala did not wish to litigate the case; Microsystems wanted the strongest legal tools possible to prevent distribution of the code. The parties agreed on a settlement in which the hackers assigned their copyright in the code to Microsystems, which could then (at least in theory) attempt to terminate any rights created by the GPL and proceed on a copyright infringement theory against anyone posting the code.²⁸⁰

When news of the settlement broke, media reports questioned whether Jansson and Skala could assign exclusive rights in CPHack to Microsystems after having placed the GPL tag on the CPHack code. One report suggested that the problem was that the GPL was not a signed writing, and therefore it could not transfer exclusive rights or did not receive priority over subsequent transfers under section 205(e) of the Copyright Act.²⁸¹ Others reported statements that rights transferred by the GPL are irrevocable.²⁸²

Suppose Jansson and Skala had assigned the rights to CPHack to an entity formed to administer GPL rights in the interests of the open-source community. What would have happened then? Microsystems no doubt would have sued the two hackers anyway. But Jansson and Skala would not have been able to assign the rights in the cp4break package to Microsystems. Microsystems might have settled for an agreement that the hackers would cease hacking its products. But if it also demanded an assignment of rights, Jansson and Skala might have found themselves in a

276. At <http://www.politechbot.com/cyberpatrol/cphack-gpl.txt> (visited Sept. 5, 2000) (on file with the *University of Illinois Law Review*). Apparently, Jansson added the tag “Released under the GPL” on his own, without consulting Skala. *Id.* The memorandum was briefly available on the Internet and probably is still available on some sites.

277. See Posting of Eddie Jansson, srm_dfr@hotmail.com, to slashdot.org (Mar. 29, 2000), at <http://slashdot.org/articles/00/03/29/0912240.shtml> (copy on file with the *University of Illinois Law Review*).

278. See Complaint, at http://www.peacefire.org/censorware/Cyber_Patrol/cp-complaint.3-15-2000.txt (last visited Aug. 25, 2000) (on file with the *University of Illinois Law Review*).

279. The restraining order eventually became a stipulated permanent injunction. It is available at www.politechbot.com/cyberpatrol/final-injunction.html (visited Apr. 6, 2000) (on file with the *University of Illinois Law Review*).

280. See Declan McCullagh, *Mattel Stays on the Offensive*, WIRED NEWS (Mar. 27, 2000), at <http://www.wired.com> (on file with the *University of Illinois Law Review*). Microsystems has not pursued this approach, however, and appeared to be satisfied with the stipulated permanent injunction. The widespread distribution of cp4break over the Internet meant that CyberPatrol needed a technical fix for its program anyway. Once it had settled with Jansson and Skala and secured its injunction, it may have seen little point in further legal proceedings.

281. McCullagh, *supra* note 272.

282. See Lawrence Lessig, *Battling Censorware*, INDUSTRY STANDARD, Apr. 3, 2000 (quoting Professor Moglen as saying that “GPL is software that cannot be revoked.”).

difficult position. They would have had to rely on the assignee to cooperate in assigning the rights, which it might or might not do, depending on its assessment of the interests of the community. This risk highlights an additional problem in the use of assignments as a signal to combat opportunism.²⁸³

Doctrinal analysis of the GNU GPL is interesting but it is not the main point. The main point of the GPL is the social structure it supports—the opportunities it creates, the practices it enables, and the practices it forbids. Contracts are social instruments, and their main effects are likely felt quite apart from litigation. Professor Llewellyn once wrote that:

The real major effect of law will be found not so much in the cases in which law officials actually intervene, nor yet in those in which such intervention is consciously contemplated as a possibility, but rather in contributing to, strengthening, stiffening attitudes toward performance as what is to be expected and what “is done.”²⁸⁴

What the GPL and other open-source licenses have done is proof of Llewellyn’s point, for the structure they support has been created and has operated thus far without judicial intervention in license disputes. This success is mostly a testament to the community members who have taken advantage of the opportunities these licenses have created. But the licenses are an elegant use of legal rules in a social context, and should be appreciated on those terms.²⁸⁵

283. The problems of assignments and termination may also compound one another. Though opportunistic termination of GPL rights may seem unlikely in ordinary cases, the chances increase dramatically if hackers such as Jansson and Skala assign exclusive rights to firms such as Microsystems, whose work rests on a conventional production model. Microsystems would likely have no qualms about terminating the rights to use its code and the right to make derivative works after the date of the assignment. A term on the original license might frustrate such attempts, but it would also make the assignment less desirable to a plaintiff such as Microsystems and therefore might make it harder for hackers such as Jansson and Skala to settle infringement cases.

284. Karl N. Llewellyn, *What Price Contract?—An Essay In Perspective*, 40 *YALE L. J.* 704, 725 n.47 (1931).

285. Indeed, one may analogize the social structure of the GPL to what Professor Williamson has termed the contract law of forbearance. See WILLIAMSON, *MECHANISMS*, *supra* note 16, at 97–100. As he points out, relations within firms are generally protected from judicial scrutiny by such doctrines as the business judgment rule. Unlike cases involving market transactions with separate firms, courts will not intervene in disputes over decisions within firms on such issues as production, the quality of goods, whether needs or output agreements have been satisfied, etc. Rules of judicial restraint therefore leave decisions within firms to the rules and norms that firms develop themselves. These rules can be justified not only on the ground that actors within firms have better information about the relevant decisions, but also on the ground that appeals to outside authorities such as judges undermine the hierarchy of the firm. The analogy may be imperfect, for open-source programmers dissatisfied with the decisions of a project maintainer probably face lower costs of leaving the project than employees do in leaving a firm. But to the extent hierarchy is valuable in coordinating complex projects, the cross-cutting nature of rights in the original code and derivative works is valuable because it makes resort to the courts impractical and thereby preserves hierarchy on the community’s terms, operating according to its norms rather than the invoked rules of copyright.

V. CONCLUSION

The open-source software movement reflects the elegant and creative use of property rights and contract law to sustain creative work based on free and transparent distribution rather than the conventional model of exclusion. The movement is a testament to the Internet's role in lowering the cost of private social ordering, thereby supporting robust social structures that were previously hobbled by the costs of social interaction. For projects that do not require costly coordination, open-source software production is likely to be robust and sustainable. More complex projects that do require coordination pose a harder question, to which a definitive answer cannot yet be given. More complex projects cannot rely solely on the lower cost of interaction on the Internet. The need for coordination implies a need for some degree of hierarchy. Hierarchy requires social structures such as norms of behavior to sustain it. The social conditions that allow these norms to form and persist are complex. Some of these conditions are generalizable, such as the general sense of open-source programmers that their work is an alternative to the "other" of conventional software production. Others, such as a desire to lessen the power of a dominant firm or to enjoy extraordinary gains in reputation through work on unusually popular projects, may not be.

As a legal matter, open-source production confirms the wonderful versatility of a system that creates general property rights and allows them to be deployed in a variety of ways. Open-source production rests ultimately on the right to exclude. But the point of the right in the open-source community is that it is not used; like the sword of Damocles, the point is not that it falls but that it hangs. This model is a striking contrast to conventional production, which rests on the very active use of exclusion. In some cases the property right may conflict with open-source tenets. This might happen, for example, if a private firm attempted to make transformative use of GPL code. In a case of true conflict and true fair use, the GPL would have to give way to the Copyright Act. But because the GPL structure seeks to create conditions favoring rapid and widespread dissemination of information, which is also the principal goal of the Copyright Act, one hopes that judges facing such a case will be sensitive to the social context in which the GPL is used and its relationship to the purposes of the Copyright Act.

The open-source movement bears watching both for what it tells us about the production of software and for what it tells us about the way our laws protect code. To the extent a right to exclude may enhance social welfare by being deployed only to facilitate the sharing of code and, in some cases, the interoperability of systems, open-source software presents a challenge to conventional thinking about the copyright reward system. The point of the system is to produce useful code at the lowest

cost and distribute it as widely and quickly as possible. The point is the code, not the reward.