OXFORD REVIEW OF ECONOMIC POLICY, VOL. 17, NO. 2

OPEN-SOURCE SOFTWARE DEVELOPMENT AND DISTRIBUTED INNOVATION

BRUCE KOGUT
Wharton School, University of Pennsylvania, and Centre de Recherche en Gestion, Ecole Polytechnique
ANCA METIU
INSEAD and Wharton School, University of Pennsylvania[1]

Open-source software development is a production model that exploits the distributed intelligence of participants in Internet communities. This model is efficient because of two related reasons: it avoids the inefficiencies of a strong intellectual property regime and it implements concurrently design and testing of software modules. The hazard of open source is that projects can 'fork' into competing versions. However, open-source communities consist of governance structures that constitutionally minimize this danger. Because open source works in a distributed environment, it presents an opportunity for developing countries to participate in frontier innovation.

I. INTRODUCTION

The central economic question posed by the Internet is how commercial investments can appropriate value from a public good called information. Information can be transported and replicated at essentially zero marginal cost and its use by one party does not preclude use by another. Since information is digitally encoded, its provision is the service that is facilitated by the Internet. Internet companies seek to collect fees and payment on information services. The difficulty in finding mechanisms to extract payments on the provision of a public good explains, to a

[1] We would like to thank the Reginald H. Jones Center, the Wharton-SMU Research Center at the Singapore Management University, and the Carnegie Bosch Institute for financing of the research. We would also like to thank Chuck Murcko, Paul David, Andrew Glyn, an anonymous referee, and participants in the *Oxford Review*'s seminar on the Internet.

large extent, the high death rate among Internet ventures.

There is, however, another side to the consequences of the economics of information. The public-good quality of information on the Internet favours the voluntary provision by users. In some cases, this provision is sustained on a long-term and ongoing basis. We can describe the participating members as forming a community to which they share allegiance and loyalty. These communities are economically interesting when they constitute not only social exchange, but also a work organization that relies upon a distributed division of labour.

A simple example of a distributed division of labour is an intranet that supports communication and work among employees in a corporation. Work can be sent back and forth, even across time zones. Teams can be physically dispersed. There are still two important background aspects to this exchange. The first is that the corporation pays the workers. The second is that the firm has the standard property rights to the product of their cooperation.

From an economic perspective, the startling aspect of open-source development of software is that people cooperate in the absence of direct pay and property right claims. Software is quintessentially an information good insofar as it can be entirely digitally encoded. In addition, its demand is influenced by its dissemination. The utility to the consumer of a given software program frequently increases with the number of users. This network externality offers, consequently, the potential for a firm to earn sustainable rents by gaining a dominant position in the market that could impede entry. However, the critical institutional feature to maintain this model is the efficacy by which intellectual property claims are upheld.

Open source means that the intellectual property rights to software code are deposited in the public domain, and hence the code can be used and changed without requiring a user fee, such as the purchase of a licence. There are thus two dimensions to open-source development: public ownership of the intellectual property and a production model by which programming work is accomplished in a distributed and dispersed community.

The recent literature on open source has focused on the economic paradox of why people contribute to a public good. Of course, this paradox is not unique to open source. Experimental economics routinely finds that people contribute more to the provision of public goods than can be predicted by self-interest (Frey and Oberholzer-Gee, 1997). The natural resolutions to this paradox are to tie provision to intrinsic rewards or to supplementary extrinsic rewards. An intrinsic reward is the satisfaction of 'helping out' as a form of gift-giving. In this view, people are altruistic because they share membership in communities that sustain reciprocity and identity. Extrinsic rewards would be the positive effect of a contribution on the reputation of a programmer, thus signalling his or her merit in a competitive job market.

These resolutions surely have a bearing upon explaining the motivations of participants, but the focus on the paradox of the provision of public goods distracts from a more far-reaching observation. The rapid growth of open-source development suggests that the traditional methods of software development are often inefficient, and these inefficiencies are permitted only due to the imposition of legal institutions to enforce intellectual property right claims. That is, firms enforce intellectual property by achieving secrecy through the organization of software production within their own organizational boundaries. Open-source development exists because, once property rights are removed from consideration, in-house production is often revealed as less efficient.

There are, then, two related hypotheses that explain open-source software.

Hypothesis one: secrecy and intellectual property create incentives that lead to behaviours that render economic activity less efficient.

These behaviours include excessive patent claims, litigation as deterrence, and the lack of access to ideas by those without ownership claims. This hypothesis is the standard assumption in economics, but is usually believed to offer the second-best solution to market failure: innovators will not innovate if they do not have patent protection. Open source challenges this theory of the second best.

Hypothesis two: the production model of open source is more efficient than in-house hierarchical models.

The central observation that leads to hypothesis two is that the concurrence in design and testing of software modules utilizes more efficiently the distributed resources connected by the Internet.

Our claim is that concerns over intellectual property create additional inefficiencies, plus prevent the deployment of more efficient production models. Once this is recognized, the interesting inquiry is to compare different open-source development models regarding their productivity and their effects on product design. We turn to this comparison after considering first the sociological background to open source.

II. COMMUNITIES OF PRACTICE

The Internet is a technological system that relies upon a communication backbone consisting largely of fibre optics and packet switching and a set of software protocols that allow for inter-operability between distributed machines and operating systems.

The other side to the Internet is its utility as a means of communication and collaborative work that predates the commercial explosion. The Internet was developed first by the US military, and then by federal programmes to create a communication network among research sites. From the start, then, the Internet was conceived as a communication mechanism for the dissemination of ideas and as a means to support distributed collaboration. The diffusion of the fundamental protocols (e.g. Transmission Control Protocol/Internet Protocol (TCP/IP), Hypertext Transfer Protocol (HTTP), Hypertext Markup Language (HTML)) arose out of research laboratories, such as CERN in Geneva. Tim Berners-Lee, who contributed the basic hypertext protocols that support the World Wide Web, noted that the Internet arose through 'webs of people' tied together through participation in research consortia (Berners-Lee and Fischetti, 1999). In other words, the Internet is not only a technology, it is also a community of developers.

The World Wide Web is an open-source software program. The property rights to these protocols lie in the public domain and anyone can access the code, that is, the written program itself. An open-source document is much like a physics experiment to which hundreds of researchers contribute.

Open-source software appears as less puzzling when its production is compared to the production of research in an academic community. Science has often been described as a conspiracy constructed to provide incentives to researchers to invest their time in the production and public dissemination of their knowledge (Dasgupta and David, 1994). To support these efforts, there are strong norms regarding the public ownership of knowledge and the importance of public validation of scientific results. Scientists are rewarded by status and prestige that can only be gained by the public dissemination of their research. In effect, the norms regarding research and its publication are aimed at rendering scientific results into a public good that can be accessed by one party without diminishing its consumption by another.

This model of scientific research conflicts strongly with the commercial interests of private enterprise to create innovations that are protected by strong intellectual property rights or by secrecy. The argument for the protection of intellectual property relies traditionally upon the theory of the second best. Society would be better off with the dissemination of innovations, but then inventors would lack the incentives to innovate. This argument is clearly at odds with the insistence in the scientific community on public access and validation of research. There is, then, a stark division between the norms that insist upon the public quality of scientific research that prevail in universities and research institutions and the concern of private enterprise to secure property rights to ideas and innovations.

Yet, many of the important contributors to the Internet and to open source were located in private enterprises. This blurring of the public and private is not unique to the Internet, but is to be found in the close networks of scientists working for biotechnology and pharmaceutical companies and other industrial research laboratories that depend upon the production of basic research. It is also to be found in the community of software developers, many of whom were employed by industrial laboratories that originally placed their software in the public domain. This clash between the private and pubic spheres is what makes the creation of the Internet such an interesting blend of economic incentives against a sociological landscape. However, there is a deeper issue involved than simply understanding these two dimensions to the historical development of the Internet. Commercial firms' insistence on private property is not only at odds with the norms of the scientific community which built the Internet, but is also at odds with an emergent model of distributed production that, for some tasks, appears far more efficient than historical alternatives.

There is, then, an important sociological aspect to understanding the origins of open-source development. Private claims to intellectual property are often seen as morally offensive owing to their distributional consequences and the fact that excluded groups are deprived of the benefits. It is fundamental in understanding the origins of open source to acknowledge the deep hostility of programmers to the privatization of software. Early software, because it was developed by monopolies such as telecommunication companies, was created in open-source environments and freely disseminated. The creators of these programs were well known in the software community. They wrote manuals, appeared at conferences, and offered help.

Earlier, we noted that the open-source community shares many of the properties of science. It is also similar to the internal labour markets that now are part of the standard economic textbook description. In their seminal analysis of internal labour markets, Doeringer and Piore (1971) noted that work was not simply the conjunction of impersonal supply and demand curves, but usually found through a matching process conducted within an organization. Critical to this matching process was the notion of skill or 'practice' by which workers gain experience specific to the firm and specific to their task. Apprenticeship often took the form of on-the-job training. The specificity of these skills drove a wedge between external and internal markets.

An operating system such as UNIX was developed in a community that spanned the boundaries of firms. To drive the wedge between the internal and external markets, AT&T chose eventually to exercise proprietary claims on its use and development. However, unlike the experience dynamic that supports internal labour markets, the expertise to develop many software programs exists in a community of practice that is wider than the boundaries of a given firm. In fact, apprenticeship in the software community consists often of learning by 'legitimate peripheral participation'. In the case of open-source software, this learning rides upon the efforts of hackers to access software code for their own use and development.[2] It is not surprising that, given this wide diversity of skills, UNIX subsequently 'forked' into a number of competing versions.

There is, then, a conflict between the external production process of software within a community and the legal governance structure that restricts development to those owning the property rights. Open source does not dissolve this distinction between the production process and the governance structure. In all open-source communities there is an explicit governance structure. The contribution made by open source is to transfer this governance structure from the firm to a non-profit body that does not own the software.

III. DESCRIPTIONS OF TWO OPEN-SOURCE MODELS

There are many software programs that are designed in open-source communities. Table 1 lists a sample of open-source projects other than Linux and Apache that we will discuss in-depth. The world of open-source software is making inroads into areas beyond operating systems, Internet and desktop applications, graphical user interfaces, and scripting languages. For example, it is also making inroads in Electronic Design Automation for Hardware Description Languages (*Linux Journal*, February 2001, p. 162). Moreover, there are now many projects designed to make open-source products more user-friendly (see Table 2).

[2] See Lave and Wenger (1991) for their discussion of 'legitimate peripheral participation'.

Table 1
Open-source Projects

Name	Definition/description
Zope	Enables teams to collaborate in the creation and management of dynamic web-based business applications such as intranets and portals
Sendmail	The most important and widely used e-mail transport software on the Internet
Mozilla	Netscape-based, open-source browser
MySQL	Open-source database
Scripting languages	
Perl	The most popular web programming language
Python	An interpreted, interactive, object-oriented programming language
PHP	A server-side HTML embedded scripting language
Other	
BIND	Provides the domain-name service for the entire Internet

Table 2
Open-source Projects Designed to Make Open-source Products more User-friendly

Name	Definition/description
KDE	Graphical desk-top environment for UNIX work-stations
GIMP	(the GNU Image Manipulation Program) Tasks such as photo retouching, image composition, and image authoring
GNOME	Desk-top environment

The commercial potential of open source rests not in the ability to charge licence fees, but in demand for consulting, support, and quality-verification services. RedHat, which sells one version of Linux, competes on the basis of customer service, and not on the basis of ownership of the intellectual property. Another example is Covalent, which is the leader in products and services for Apache, and the only source of full commercial support for the Apache Web server.[3]

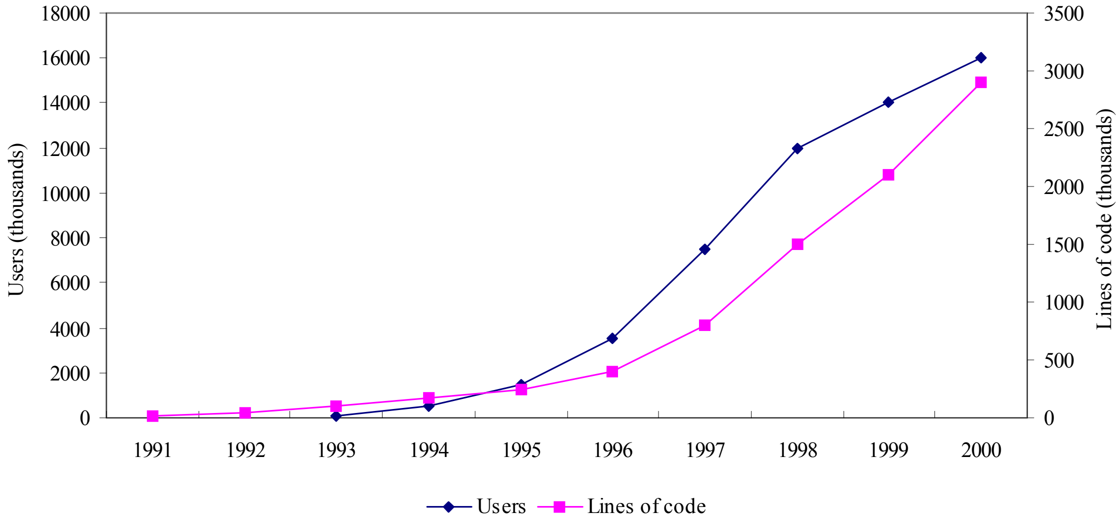
Linux and Apache are two of the most successful open-source software communities (the World Wide Web is obviously a third.) To understand better how open source works, and how the various communities differ, we provide a short description of both.

(i) Linux

Linux is a UNIX operating system that was developed by Linus Torvalds and a loosely knit community of programmers across the Internet. The name Linux comes from Linus's UNIX. In 1991, Linus Torvalds, a Finnish computer science student, wrote the first version of a UNIX kernel for his own use. Instead of securing property rights to his invention, he posted the code on the Internet with a request to other programmers to help upgrade it into a working system. The response was overwhelming. What began as a student's pet project rapidly developed into a non-trivial, operating-system kernel. This accomplishment was possible because, at the time, there already existed a large community of UNIX

[3] For a more comprehensive list of companies, see Krueger in *Wired* magazine, 7 May 1999, available at http://www.wired.com/wired/archive/7.05/tour.html

Figure 1
Growth of Linux: Number of Users and Number of Lines of Code



Sources: RedHat Software, Inc. and *Forbes*, 10 August 1998.

developers who were disenchanted by the fact that vendors had taken over UNIX development. They also were unhappy with the growing reliance on Microsoft's proprietary server software.

The Linux development model is built around Torvalds's authority, described by some as 'benevolently' exercised.[4] Legally, anyone can build an alternative community to develop other versions of Linux. In practice, the development process is *centralized*, being distributed but subject to hierarchical control. New code is submitted to Torvalds, who decides whether or not to accept it, or request modifications before adding it to the Linux kernel. In this sense, Torvalds is the undisputed leader of the project, but there is no official organization that institutionalizes this role. As Linux grew in popularity and size, Torvalds became overwhelmed with the amount of code submitted to the kernel. As Linux members noticed, 'Linus doesn't scale.' Therefore, Torvalds delegated large components to several of his trusted 'lieutenants' who further delegated to a handful of 'area' owners. Nowadays, several developers have more-or-less control over their particular subsections. There is a networking chief, a driver chief, and so forth. While Torvalds has ultimate authority, he seldom rejects a decision made by one of these sub-administrators.

Torvalds accumulates the patches received, and then releases a new monolithic kernel incorporating them. For software that does not go into the kernel, Torvalds does not prevent others from adding specialized features. These patches allow even greater customization without risking the integrity of the operating system for the vast majority. Sometimes optimizing for one kind of hardware damages the efficiency for other hardware. Some users require 'paranoid security' that, by definition, cannot be useful if disseminated; or, some incremental innovations are too experimental to inflict on everyone.

The number of contributors also grew dramatically over the years, from Linus Torvalds in 1991 to 10,000 developers in 1998 (*Forbes*, 10 August, 1998). Figure 1 portrays the remarkable growth in the number of Linux users (16m in 2000) and in the product's lines of code (2.9m in 2000). In terms of server operating systems shipped, Linux's market share was 24 per cent in 1999. According to International Data Corporation, over the next 4 years, Linux shipments will grow at a rate of 28 per cent, from 1.3m in 1999 to 4.7m in 2004.[5]

[4] See an interview with Brian Behlendorf: http://www.linux-mag.com/2000-04/behlendorf_01.html. Even Torvalds views himself as governing by his acknowledged software expertise and skills as a project manager (see Appendix to DiBona *et al.*, 1999.)
[5] See IDC report at http://www.idc.com/itforecaster/itf20000808.stm

(ii) Apache

The Apache HTTP server project is a web server originally based on the open-source server from the National Center for Supercomputing Applications (NCSA). A web server is a program that serves the files that form web pages to web users (whose computers contain HTTP clients that forward their requests). Web servers use the client/server model and the World Wide Web's Hypertext Transfer Protocol. Every computer on the Internet that contains a web site must have a web-server program. The name reflects the practice of university-laboratory software being 'patched' with new features and fixes ('a patchy server').

The project was started in 1995 to fix an NCSA program. For most of its existence, there have been fewer than two dozen people seriously working on the software at any one time. The original group included eight people who later became known as webmasters, and many who went on to start open-source projects at commercial enterprises. Several of the original members came from the University of Illinois, which also spawned the web browser that became Netscape. The original group constituted the Apache core, and is responsible for the primary development of the Apache HTTP server.

The development for the Apache model is *federal*, based upon a meritocratic selection process. While access to the source code and the history information of changes is available to anyone, the ability to make changes is reserved for the Apache board, comprised of people that have been chosen because of proven ability and past contributions. Other contributors to Apache can join three different groups. The developer e-mail list consists of technical discussions, proposed changes, and automatic notification about code changes, and can consist of several hundred messages a day. The Current Version Control archive consists of modification requests that resulted in a change to code or documentation. There is also the problem-reporting database in the form of a Usenet that is the most accessible list, consisting of messages reporting problems and seeking help.

The coordination of the development process is achieved via two types of rules. The initial rule, called 'review-then-commit' (RTC), was used during 1996 and 1997. It states that in order for a change to master sources to be made, a submitted patch would first need to be tested by other developers who would apply it to their systems. This rule leads to a time-consuming process, and it does not encourage innovation. Therefore, in 1998 a new process was introduced, the 'commit-then-review' (CTR). CTR speeds up development while exercising quality control. However, it demands vigilance on the part of the development team. Controversial changes need first to be discussed on the mailing list.

Mailing-list discussions typically achieve consensus on changes that are submitted. However, particularly controversial topics may call for a vote. Because Apache is a meritocracy, even though all mailing-list subscribers can express an opinion by voting, their actions may be ignored unless they are recognized as serious contributors.

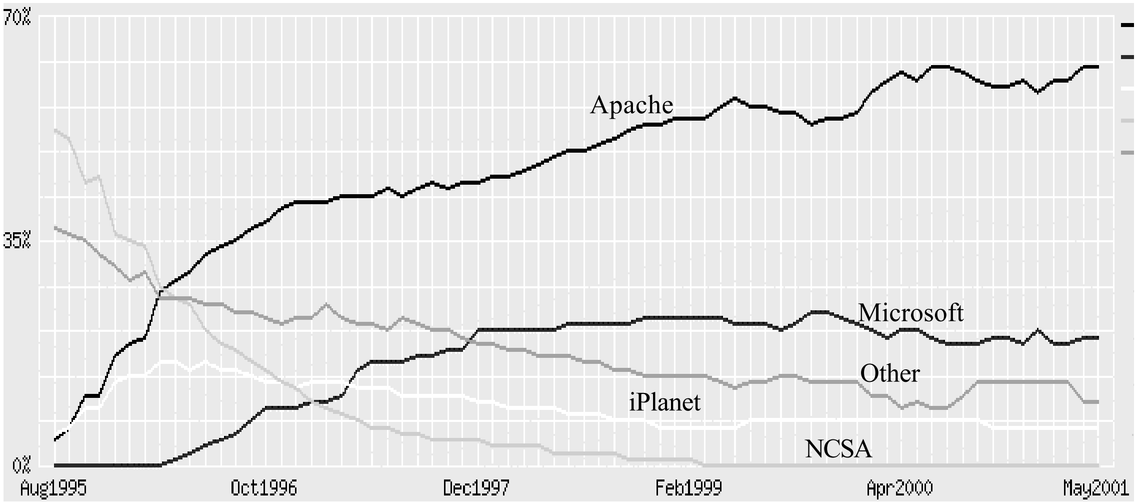
New versions of Apache are released when developers achieve consensus that it is 'ready', and not by set calendar dates. Someone volunteers to be the release manager, who then receives 'code ownership' (Mockus *et al*., 2000). The developer has the responsibility for getting agreement on the release schedule, ensuring that new commits are not too controversial, contacting the testers' mailing lists, and building the release. Once a release is out, people start hacking on it.

Apache has a 62 per cent share of the Internet server market (see http://www.netcraft.co.uk/survey/). Figure 2 graphs Apache's steadily increasing market share, beating out proprietary products such as Netscape's and Microsoft's server suites. Apache is now an umbrella for a suite of projects such as XML and Java projects.

IV. INTELLECTUAL PROPERTY AND LICENCES

The various open-source licences share the fundamental trait that the property rights to their use are placed in the public domain. They differ in the extent to which they allow public-domain property to be mixed with private property rights. The historical trends have been to tolerate a hybrid of both. As noted earlier, these issues are similar to the conflicts surrounding public and private property claims to the

Figure 2
Growth of Apache's Market Share



Source: Netcraft Survey.

results of basic research funded by public research institutions.

The first open-source licence was Richard Stallman's General Public License (GPL) created for the protection of the GNU operating system.[6] It was the decision of AT&T to issue proprietary control over UNIX that led Stallman to start the GNU Project in 1984 to develop a complete UNIX-like operating system as free software. Stallman started the Free Software Foundation (FSF) to carry out this project, and called his licence 'copyleft' because it preserves the users' right to copy the software.[7]

As commercial enterprises started to take note of open-source software, some members of the community thought they needed to sustain this interest by toning down the free software rhetoric. On the basis of the licence for the Debian GNU/Linux distribution developed by Bruce Perens in 1997, the Open Source Definition (OSD) was born.[8] This licence differs from the GPL. The GPL forces every program that contains a free software component to be released in its entirety as free software. In this sense, it forces 'viral' compliance. The OSD only requires that a free/open-source licence allow distribution in source code as well as compiled form. The licence may not require a royalty or other fee for such a sale. Consistent with the requirements of the OSD, the Berkeley System Distribution (BSD) and Apache licences allow programmers to take their modifications private, i.e. to sell versions of the program without distributing the source code of the modifications.

The boundaries between the public and private segments of the software developed by the open-source community are thus not distinct. Even under the GPL, which allows double licensing, it is possible to make money on the commercial version. An author can release the source code of a project under an open-source licence, while at the same time selling the same product under a commercial licence. It is also possible to make money by developing proprietary applications for open-source infrastructure. Applications that operate independently (e.g. leaf notes in the software tree) can be proprietary, but the infrastructure should be open source.

The open-source licences conflict with most, but not all, interpretations of the functioning of a patent system. Mazzoleni and Nelson (1998) note recently that patent law serves several competing functions.

[6] GNU is a recursive acronym for 'GNU's Not UNIX's', and it is pronounced 'guh-NEW'.
[7] For these efforts, Stallman was called 'the last hacker' in a book on the beginnings of the computer (Levy, 1984).
[8] For more details, see http://www.debian.org/social_contract.html#guidelines

The first function is the resolution of market failure to reward innovators. Since inventions can be copied, the inventor requires an enforceable property claim in order to have a temporary monopoly to extract an economic return. But patents also serve other functions. They place in the public domain the knowledge of the invention and they establish property rights to important 'gateway technologies' that permit the further development of derived inventions in an orderly way. The critical feature of these arguments is the observation that research is an input and a product. By protecting the product, the danger is to slow progress by restricting the use of the innovation as an input into subsequent efforts.

The modern economics and law tradition in property rights has argued that patents are a solution to the 'tragedy of the commons' problem. In a seminal article, Hardin (1968) argued that public goods are prone to be overused when too many owners have the right to use them, and no owner has the right to exclude another. Hardin's explanation has also fuelled the policy of privatizing commons property either through private arrangements (Ostrom, 1990) or the patenting of scientific discoveries.

More recent legal writings have, however, questioned this tradition. Heller and Eisenberg (1998) have pointed out that public goods are prone to under-use in a 'tragedy of the anticommons' when too many individuals have rights of exclusion of a scarce resource. An example of under-use of a scarce source is the fragmentation of rights in biomedical research in the USA. The need to access multiple patented inputs may deter a user from developing a useful product.

In recent years, there has been considerable attention paid to the cost of excessively strong property regimes by which to reward innovation. In particular, the recent expansion of the legal protection of software from copyright to patents has been decried as a threat to innovation and to the sharing of knowledge in fast-paced industries. David (2000) has noted the dangers of this type of encroachment in the commons of academic research. Similar problems arise in other industries. Lerner (1995), for example, found that patents by large firms in biotechnology have effectively deterred smaller firms from innovating in these areas. In other words, the shortcomings of the patent-awarding process defeat the stated purpose of the patent system to provide incentives to innovate. Because firms use the legal system strategically, the second-best argument for patent protection becomes less clear. The implication is that no protection might, in some cases, dominate the policy of providing monopoly rights to the exercise of a patent.

American law has permitted the violation of private property if the loss of public access is considered to be excessive. Merges (1999*a*) cites the case of litigation over the right to built a new bridge over the Charles River in Boston. In 1837, the courts ruled in favour of the right of public access and against a company that had secured an exclusive franchise to operate bridges over the Charles River. Similarly, courts have rarely upheld the claims of companies to deter former employees from exploiting an idea. Hyde's (1998) study of the Silicon Valley shows that the 'law in action' in the region encourages rapid diffusion of information by protecting start-ups and employee departures.

Various solutions to the fragmentation of rights have been proposed in recognition of the fact that ownership and control of the cornerstone pieces on which the digital economy is built are crucial issues for economic policy. To meet the need of inter-operability among standards, Merges proposed patent pools as solutions that reduce the volume of licensing and lead to greater technological integration (Merges, 1999*b*). Recently approved pools, such as the MPEG-2 pool that brings together complementary inputs in the form of 27 patents from nine firms, could serve as a guide to for other industries. The pool was an institutional expression of the creation of the MPEG-2 video compression technology standard. Patent-holders license their MPEG-2 patents to a central administrative entity that administers the pool on behalf of its members. The pool includes only essential patents, i.e. those patents required to implement a widely accepted technological standard. Patent pools suffer, however, from a number of problems, the most important one being the potential for a hold-up by one of the parties.

A general patent licence avoids this potential by a 'viral' quality to enforce compliance. The GPL is unique in its provision that does not allow programmers to take modifications private. This 'viral' clause results in all software that incorporates GPL-ed programs becoming open source as well. As noted above, patents serve two different functions: to incite innovation and to encourage incremental exploration. Public licences, such as the GPL, permit this additional function to operate, while obviating the negative consequences of a second-best policy. Since anyone has the right to use and modify an open-source software program, these licences provide maximum freedom for the exploitation of incremental innovations.

It is, however, the more pragmatic licences that support Apache that pose a danger to the incentives to form open-source projects. The licences that permit a blending of open and proprietary code put at risk the ideals on which the community has been built. For open-source contributors and advocates, intellectual property is a commons that needs to be protected from enclosure.[9] As such, open source provides an answer to the fragmentation of protected—patented or copyrighted—knowledge. Moreover, the open-source licences allow the community to protect the code it produces and to induce compliance with the philosophy expressed in these licences. It is these licences that keep the code in the commons, and they protect the generalized reciprocity that characterizes the community culture.

(i) Governance Structure

However, the licences may not be enough by themselves. The question then is whether there are in place governance structures that will prevent fragmentation of code into proprietary islands. Lawrence Lessig (1999) made an important argument that software code contains the constitutional rules by which participants behave in virtual communities, such as chat rooms. For open-source development, the causality runs the other way. The different governance structures influence the development of the code in at least two important ways. The first is that every open-source software program runs the danger of 'forking', as seen in the case of UNIX or in Java. The second is that organization by which work is delegated influences the product design.

Neither Linux nor Apache has forked into competing versions. The Apache governance structure has the advantage of being coalitional. The coalition itself can change, as participating developers can migrate to more important roles depending upon their contribution. It is thus easier to organize a response to potential efforts to 'fork' the code. The Linux community is also hierarchical, as we saw, but highly centralized on Torvalds. If Torvalds himself should play a less central role and hence the role of the charismatic leader (in Weber's sense) fades, then the methods by which disputes would be resolved are not at all obvious.

There is the interesting issue of whether the design of the product itself can force compliance. For example, initially Torvalds wrote the kernel as an integral unit, contrary to academic opinion. However, over time, it too became more modular. Apache, by virtue of its coalitional structure, from the start was very modular. There is thus convergence in product design, though the initial structure of the products reflected the differences in the governance structures of the two communities. A question, largely unexplored, is whether the vestiges of the Linux design force agreement on the interfaces between the modules and the kernel and core modules. In this case, the choice of technological design might force compliance to a standard. However, as this possibility is not compelling, it is unsurprising that Linux should be protected under a GPL that requires all code to be non-proprietary. In this way, if the code should be balkanized, it will not at least be proprietary. The Apache licence does not prevent proprietary code from being mixed with open source, but it also has a stronger governance structure to respond to defectors.

In conclusion, the open-source licences generally have the advantage of forcing the code into the public domain. They thereby favour a dynamic by which incremental innovations can be rapidly contributed to improve the code and to add functions. The danger of open-source development is the potential for fragmenting the design into competing versions. Governance structures offer some potential for preventing 'forking', as well as technological choices that might force compliance.

[9] See interview with Tim O'Reilly in the *Linux Journal*, February 2001.

V. THE SOFTWARE PRODUCTION PROCESS

The second argument for open-source software is that it offers a better model for development. There is an active debate in the software literature regarding how much software development is 'craft' and how much is 'routinized'. The craft nature of software development was strained by the demand for 'integral' programs that required thousands of engineers. Brooks (1975) documented the difficulties posed by the creation of the operating system for the IBM 360 large-frame computer. A major problem for the design of sophisticated software programs has been reducing the complexity in the development process.

In traditional software production processes, two fundamental contributions have sought to reduce this complexity. The first is the use of 'hidden knowledge' incorporated in a module, with team managers focusing on interfaces to optimize overall functionality. The second contribution has been the partitioning of software development into discrete steps that can be conducted sequentially or concurrently. Both of these principles are used in open-source development.

While similar principles of design are used, open source benefits from two sources of efficiency gain. The first is the efficiency of implementing production in a distributed community of practice that permits frontier users also to be contributors. This gain is especially striking in light of von Hippel's finding that many innovations originate with users, not producers (von Hippel, 1988). The second source is concurrent debugging and design. Whereas it is standard practice for software houses to release beta versions of their products, the release of open-source code permits a 'tweaking' of the code on a decentralized basis that can then be incorporated into official releases. It would be helpful to look at both of these sources of efficiency gains in more detail.

(i) User Motivation

The critical question posed by an open-source licence is whether there are sufficient incentives for developers to contribute effort to innovation. One claim is that developers contribute out of a sense of 'altruism'. Indeed, there is considerable evidence in economic behaviour that people ignore economic calculations in their decisions. For example, the experiments by Kahneman *et al*. (1986) and Bies *et al*. (1993) pointed to the role of fairness, by which people share a fixed reward. People also defect less than the prediction on prisoner-dilemma games. Defection falls dramatically with communication and with very simple screening devices (Ostrom *et al*., 1992; Frank, 1988). The importance of reciprocity in the exchanges among members of the open-source community has been recently documented by Lakhani and von Hippel (2000). Their study of the support groups for the Apache web server shows that the most important reason for people posting answers on Usenet groups is the desire to help because they have been helped by others, or because they expect to need others' expertise in the future.

Lerner and Tirole (2000) propose an explanation that does not rely upon altruism, or identity. They argue that contribution to an open-source project is much like a tournament that signals merit. Contributors enjoy improved prospects in the labour market by signalling their merit.

These two perspectives of gift-giving and labour-market signalling reflect two different views of motivation. Gift-giving reflects an 'intrinsic' motivation, whereby the individual finds reward in the public validation of his or her value. Labour-market signalling is an 'extrinsic' motivation that ties contribution to pecuniary reward. Both motives may in fact be operating, though it would seem that communities with mixed motivations often dissolve owing to obvious free-rider problems. Indeed, many labour-supply models have noted the problem of signalling false quality. An efficient wage is one of many devices suggested to attain effort and quality from workers when defection is probabilistic.

The importance of distinguishing between these two motivations is central to the classic study on gift-giving in which Titmuss (1971) found that the extrinsic reward of paying for blood expands the supply but also makes voluntary contributions less intrinsically rewarding. The consequence is, ironically, the potential destruction of the market for blood by increasing uncertainty over quality.[10] These much-discussed results have two implications. The first is that the donors themselves have the best knowledge of the likely quality of their blood. Given the existing technologies and this information asymmetry, it makes sense to reduce potentially the blood supply but gain a 'free' filter to be imposed by the donor that leads to an overall higher-quality supply. The second is that the donor is also a potential recipient. In other words, a voluntary policy provides a highly motivated donor.

There is little evidence that open-source participants are more motivated. Indeed, this conclusion would appear to be hard to defend on the existing evidence, especially if effort must be acquired for less interesting projects. However, the more relevant deduction is that the open-source model relies upon knowledgeable users to contribute as developers. It is not the average motivation that may matter, but rather the attraction of highly motivated and capable talent to the project. In this sense, open source more effectively exploits the intelligence in the distributed system.

(ii) Concurrence of Debugging and Code-writing

We claim that open source exploits the intelligence in the distributed system. The development of complex software products poses severe engineering and managerial difficulties. To meet the challenge of reducing the costs of producing complex software, many companies adopted structured approaches to software development. Cusumano's study of the 'software factory' documents how software design moved from art to routinized tasks manipulating standardized modules (Cusumano, 1991). This approach culminated in an attempt to rationalize the entire cycle of software production, installation, and maintenance through the establishment of factory-like procedures and processes.

The factory production process is not, however, well suited to all software design processes. Glass (1995) views software as a creative enterprise that cannot be fully routinized. Methodologies to convert design into a disciplined activity are not suited to addressing new problems to be solved (1995, p. 41). At the same time, writing of code involves solving the detail-level problems left unsolved in an inevitably incomplete design.

The factory approach to software development applies the Babbage principle of the mental division of labour. In this model, intelligent work is specialized to the design group, code writing is given to a less skilled group, and debugging and maintenance to an even less skilled group. *A reasonable production function for this kind of process is a 'weak link' chain, where the least productive element in the process determines the output* (see Becker and Murphy, 1992, for an example).

The interactive approach suggests a production function in which value is maximized, subject to the constraints of threshold quality and time to market. This process will be less structured than a 'waterfall' sequence, where the design stage precedes coding and testing, but will allow for concurrent design and implementation. *This model suggests that the software production is as good as its most productive member.* It is in this sense that open source exploits the intelligence in the community; it provides a matching between competence and task.

Open-source development permits this resolution of complexity by consistently applying the principles of modular design. The modularization of software evolves through a series of complex adaptations. Open source has several traits in common with the description by Baldwin and Clark (2000) of the recombinative evolution of the assembly of component modules of computers. By relying upon an external market that proposes incremental module improvements, computer assemblers benefit from the distributed intelligence of competing suppliers. It is not surprising that some have taken this to be the key element to open-source development. For example, Axelrod and Cohen (2000) explicitly treat Linux as an example of a complex adaptive system. The open-source licences permit distributed and uncoordinated developers to propose variants to the existing program. These variants are then submitted to a selection process that chooses the better-performing program.

[10] The findings that monetary rewards can have a negative effect on motivation are not new—see Lepper and Greene (1978) and, more recently, Gneezy and Rustichini (2000).

Table 3
Contributions to Linux and Apache

Contributions/person	Linux contributors	Apache contributors
1	1,866	232
2	355	36
3	110	16
4	44	9
5	17	5
6	12	2
7	2	5
8	5	
9	5	2
10–19	9	14
20–30	4	5
Total	2,429	326

Source: Our analyses of Apache, and Dempsey *et al.*'s (1999) study of a subset of the Linux community.

The complex-adaptive-system approach captures the advantage of using system testing in a distributed community. However, the community is far more hierarchically organized for the actual development of software code than suggested by the metaphor of a population of interacting agents. For the contribution of large modules, Apache and Linux both assign these tasks to developers who manage the project.

It is not surprising that, in spite of the large number of participants in open-source communities, the actual number of constant contributors is small. We analysed the 'Changes' files to Apache between March 1995 and February 2000. These files list the new patches included in each new version of Apache, as well as their author. The analysis reveals that a small number of developers are responsible for the majority of contributions. While there were 326 people who contributed patches during the analysed period, most of these individuals—232 to be precise—only contributed one patch per person, and 36 only two patches per person. In contrast, the top five contributors each made between 20 and 30 changes, and another 14 individuals each made between ten and 19 changes. Other researchers have obtained similar results. Mockus *et al*. (2000) found that the top 15 Apache developers contributed more than 83 per cent of the basic changes, and that the changes done by core developers are substantially larger than those done by the non-core group. The role of system tester is the function reserved primarily for the wide community of Apache users. The same pattern of contributions also holds in the Linux community. Table 3 shows the frequency count of the changes from Apache and from a study on a subset of the Linux community (see Dempsey *et al.*, 1999).

Hence, it is not modularity that gives open source a distinctive source of advantage, because it too relies on hierarchical development. Rather the source of its advantage lies in concurrence of development and debugging. In spite of its unglamorous nature, maintenance alone can represent anywhere between 50 and 80 per cent of the average software budget (Yourdon, 1996). The largest part of the developer community are not involved with code writing, but with code debugging.

Raymond (1998) has eloquently summarized the efficiency of the open-source model in debugging code: 'given enough eyeballs, all bugs are shallow'. Such claims have been substantiated by researchers who compared the performance of commercial and open projects in terms of the speed of debugging. Kuan (2000) found that open-source projects ranked higher on the debugging dimension than closed-source projects. Also, Mockus *et al*. (2000, p. 6) found that the productivity of Apache development is very high compared to commercial projects, with lean code and lower defect density even before system test.

The efficiency of the open-source development model is indirectly established by software firms' efforts to emulate it, without even realizing it. Cusumano and Selby (1995) explain that in order to encourage exchange of ideas, Microsoft builds software teams and cultivates developer networks within the company. In this sense, Microsoft creates an internal community to appraise and debug the innovations of software teams. Yourdon (1996) also notes the company's practice of instituting the 'push back method' whereby people challenge each other's ideas.

Yet, this simulated 'open-source' environment differs not only in size, but also by separating final users from the process. One of the most important contributions by open source is, by releasing the code, to let users themselves fix the bugs. As often noted, no one knows the number of bugs in a Microsoft product, because the software is proprietary. By placing the code in the public domain, open-source development corrects bugs concurrently with design and implementation. Users participate usually by posting questions and complaints through 'usenets'. This activity is separate from the design activity that, as explained above, remains hierarchically organized.

(iii) When Will We Not See Open Source

Of course, not all software projects are amenable to open-source development. An operating system, because it is long-lasting and widespread, can benefit from a system that provides rapid improvement and has a low catastrophic risk; but for example, a software system that is tailored to supporting trading activity on a specific stock market is an unlikely candidate for open sourcing; the code is too specific and hence not reusable and the catastrophic risk is too high.

A product that is not modular would also not be appropriate for open-source development. A molecule, for example, is not modular; changing atoms drastically alters its pharmaceutical properties. Modularity can be achieved by breaking up the discovery and trial sequence into many steps. But such steps cannot be done concurrently, so there is no gain to open-source debugging.

Thus the range of modular to integral will greatly influence the application of open-source development, as well as the speed of the development cycle. For products that are modular and for which development times are short, community development by open source offers clear advantages. The important issue is whether the weak appropriability of open-source development swings the choice towards less efficient proprietary models of development that have strong intellectual property mechanisms by which to appropriate rents to innovation.

VI. CONCLUSIONS ON ITS ECONOMIC POTENTIAL

As a back of the envelope exercise, it is interesting to ask whether open source might make any material impact on developing countries. Developing countries present two central features. They have, in aggregate, the bulk of the world population and, hence, of the world's brain power. Yet, they have a minuscule share of world technological innovation. This disequilibrium has surely been a potent force in explaining the migration of educated individuals from poor to rich countries.

Can open source provide an alternative model whereby innovation can occur on a more distributed basis? Over the past 10 years, the Indian software industry has grown at annual rates of over 50 per cent. The industry's revenue in the fiscal year 1999/2000 was $5.7 billion. The most prominent centre of software development in India is Bangalore, which accounted for over a quarter of India's total software exports in 1999/2000. In 1991, the government lowered tariffs on foreign goods and loosened investment restrictions. The Indian success in capitalizing on this liberalization is aided by the large and highly educated work-force of engineers. India produces about 70,000–85,000 software engineers annually, along with about 45,000 other IT graduates. The government plans to double the intake of IT graduates for the 2001–2 academic year.

The Indian industry is not large relative to total GNP or to total employment. As low value-added links in the global software production chain, it would take rather improbable multipliers on the domestic economy to lead to the expectation that they could be engines for growth. Yet, if the value added in their software contribution should increase, then a more dynamic scenario is feasible. The critical question is whether Indian companies can move from 'body-shop' services to high value-added innovations. Can a developing country join the frontier of innovation or is it trapped in the seesaw of low value-added exports of products and high value-added exports of human capital to developed countries?

In this regard, open source poses a body of policy questions that are familiar to economists and policy-makers regarding competition regulation and intellectual property rights. The traditional dilemma for anti-trust regulation and law is the balance between providing incentives for innovation by allowing for monopoly profits and yet avoiding the foreclosure of access to intellectual property that serves as a complement to other innovative endeavours. The doctrine of essential facility has sought to address this dilemma.[11] Software operating systems, such as Microsoft's Windows, are examples of such essential facilities. The regulatory and legal solution to treating essential facilities has been to respect the monopoly profits associated with the primary market, but to seek to require access for the purpose of entry into new markets. However, this solution is cumbersome and expensive; it addresses the issue of intellectual property rights without acknowledging that it would not permit the operation of the production model of open source.

The primary policy implication of open source is to emphasize that excessive intellectual property regimes prevent the implementation of production models that the Internet makes far more feasible today than previously. As we noted earlier, the open-source community shares many properties with the conduct of research by scientists. It is ironic that the trend in intellectual property of research content has been towards privatization of the commons, such as in the commercialization of data services and interference with 'fair use' under copyright law through digital signatures (see David, 2000). However, the loss to the academic community is not simply the exclusion due to limited financial resources to purchase these data. The loss is also the erosion of communal values that undergird an open production model for research that is impressively efficient in creating incentives for individuals to invest in research *and* its dissemination.

The Internet is itself the outcome of a combination of public and private incentives. Since its utility is the provision of information goods that are both outputs and inputs for further innovation and research, the dissemination of information is desirable. Consequently, the Internet is bound to be marked by deep conflicts over the intellectual property as a right granted by legal discretion versus entitlement. The experience of open source poses a fundamental challenge to the traditional concerns over the effects of weak intellectual property 'rights' on innovation by representing an endogenous mechanism of global innovation that offers an efficient production model. In a time of deepening disparities in world incomes and massive migration flows of educated people from poor to rich countries, this alternative model deserves close attention in changing the tone of the debate. Open source represents the emergence of a production model ideally suited for the properties of information that can be digitally coded and distributed. The test of a contest of intellectual property over information goods must consider the economic loss of impeding the distributed organization and production of knowledge on a global basis.

[11] We thank a referee for this discussion.

REFERENCES

Axelrod, R., and Cohen, M. (2000), *Harnessing Complexity: Organizational Implications of a Scientific Frontier*, New York, Free Press.

Baldwin, C., and Clark, K. (2000), *Design Rules. Vol. 1: The Power of Modularity*, Cambridge, MA, MIT Press.

Becker, G. S., and Murphy, K. M. (1992), 'The Division of Labor, Coordination Costs, and Knowledge', *The Quarterly Journal of Economics*, **107**, 1137–60.

Berners-Lee, T., and Fischetti, M. (1999), *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*, San Francisco, CA, Harper.

Bies, R., Tripp, T., and Neale, M. (1993), 'Procedural Fairness and Profit Seeking: The Perceived Legitimacy of Market Exploitation', *Journal of Behavioral Decision Making*, **6**, 243–56.

Brooks, F. P., Jr (1975), *The Mythical Man-Month*, Reading, MA, Addison-Wesley.

Cusumano, M. A. (1991), *Japan's Software Factories*, New York, Oxford University Press.

— Selby, R. W. (1995), *Microsoft Secrets*, New York, Free Press/Simon & Schuster.

Dasgupta, P., and David, P. (1994), 'Towards a New Economics of Science', *Research Policy*, **23**, 487–521.

David, P. (2000), 'A Tragedy of the Public Knowledge "Commons"? Global Science, Intellectual Property, and the Digital Technology Boomerang', All Souls College, Oxford, mimeo.

Dempsey, B. J., Weiss, D., Jones, P., and Greenberg, J. (1999), 'A Quantitative Profile of a Community of Open Source Linux Developers', School of Information and Library Science at the University of North Carolina at Chapel Hill. http://metalab.unc.edu/osrt/develpro.html

DiBona, C., Ockman, S., and Stone, M. (eds) (1999), *Open Sources: Voices from the Open Source Revolution*, Sebastopol, CA, O'Reilly.

Doeringer, P. B., and Piore, M. J. (1971), *Internal Labor Markets and Manpower Analysis*, Lexington, MA, Heath.

Frank, R. (1988), *Passions Within Reason: The Strategic Role of the Emotions*, New York, W. W. Norton & Co.

Frey, B. S., and Oberholzer-Gee, F. (1997), 'The Cost of Price Incentives: An Empirical Analysis of Motivation Crowding-out', *American Economic Review*, **87**(4), 746–55.

Glass, R. L. (1995), *Software Creativity*, Englewood Cliffs, NJ, Prentice Hall.

Gneezy, U., and Rustichini, A. (2000), 'Pay Enough or Don't Pay At All', *Quarterly Journal of Economics*, **115**(3), 791–810.

Hardin, G. (1968), 'The Tragedy of the Commons', *Science*, **162**, 1243–8.

Heller, M. A., and Eisenberg, R. S. (1998), 'Can Patents Deter Innovation? The Anticommons in Biomedical Research', *Science*, **2480**, 698–701.

Hyde, A. (1998), 'Silicon Valley's Efficient Abolition of Trade Secrets', in *Corporate Governance Today*, Columbia Law School.

Kahneman, D., Knetsch, J. L., and Thaler, R. H. (1986), 'Fairness as a Constraint on Profit Seeking: Entitlement in the Market', *American Economic Review*, **76**, 728–41.

Kuan, J. (2000), 'Open Source Software as Consumer Integration into Production', Working Paper, Berkeley.

Lakhani, K., and von Hippel, E. (2000), 'How Open Source Software Works: "Free" User-to-user Assistance', Working Paper No. 4117, Cambridge, MA, MIT Sloan School of Management.

Lave, J., and Wenger, E. (1991), *Situated Learning: Legitimate Peripheral Participation*, Cambridge, Cambridge University Press.

Lepper, M. R., and Greene, D. (eds) (1978), *The Hidden Costs of Reward: New Perspectives of the Psychology of Human Motivation*, Hillsdale, NJ, L. Erlbaum Associates.

Lerner, J. (1995), 'Patenting in the Shadow of Competitors', *Journal of Law and Economics*, **38**(2), 463–95.

— Tirole, J. (2000), 'The Simple Economics of Open Source', NBER Working Paper 7600.

Lessig, L. (1999), *Code and Other Laws of Cyberspace*, New York, Basic Books.

Levy, S. (1984), *Hackers: Heroes of the Computer Revolution*, New York, Dell.

Mazzoleni, R., and Nelson, R. R. (1998), 'Economic Theories about the Benefits and Costs of Patents', *Journal of Economic Issues*, **32**(4), 1031–52.

Merges, R. P. (1999*a*), 'Who Owns the Charles Bridge? Intellectual Property and Competition in the Software Industry', Working Paper, Berkeley.

— (1999*b*), 'Institutions for Intellectual Property Transactions: The Case of Patent Pools', Working Paper, Berkeley.

Mockus, A., Fielding, R. T., and Herbsleb, J. (2000), 'A Case Study of Open Source Software Development: The Apache Server', *Proceedings of the 22nd International Conference on Software Engineering*, 263–72.

Ostrom, E. (1990), *Governing the Commons: The Evolution of Institutions for Collective Action*, Cambridge, Cambridge University Press.
— Walker, J., Gardner, R. (1992), 'Covenants With and Without a Sword: Self-governance Is Possible', *American Political Science Review*, **86**(2), 404–17.
Raymond, E. S. (1998), 'The Cathedral and the Bazaar', http://www.tuxedo.org/~esr/writings/cathedral-bazaar/
Titmuss, R. M. (1971), *The Gift Relationship: From Human Blood to Social Policy*, New York, Pantheon.
Von Hippel, E. (1988), *The Sources of Innovation*, New York, Oxford University Press.
Yourdon, E. (1996), *Rise and Resurrection of the American Programmer*, Upper Saddle River, NJ, Prentice Hall.