Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science

Eric von Hippel • Georg von Krogh
Sloan School of Management, Massachusetts Institute of Technology, 50 Memorial Drive, Cambridge, Massachusetts 02141
Institute of Management, University of St. Gallen, Dufourstrasse 48, CH-9010 St. Gallen, Switzerland
evhippel@mit.edu • georg.vonkrogh@unisg.ch

Abstract

Currently, two models of innovation are prevalent in organization science. The "private investment" model assumes returns to the innovator result from private goods and efficient regimes of intellectual property protection. The "collective action" model assumes that under conditions of market failure, innovators collaborate in order to produce a public good. The phenomenon of open source software development shows that users program to solve their own as well as shared technical problems, and freely reveal their innovations without appropriating private returns from selling the software. In this paper, we propose that open source software development is an exemplar of a compound "private-collective" model of innovation that contains elements of both the private investment and the collective action models and can offer society the "best of both worlds" under many conditions. We describe a new set of research questions this model raises for scholars in organization science. We offer some details regarding the types of data available for open source projects in order to ease access for researchers who are unfamiliar with these, and also offer some advice on conducting empirical studies on open source software development processes.

(*Open Source Software*; *Innovation*; *Incentives*; *User Innovation, Users, Collective Action*)

1. History and Characteristics of Open Source Software Development Projects

Open source software is software that is made freely available to all. Open source software development projects are Internet-based communities of software developers who voluntarily collaborate to develop software that they or their organizations need. Open source projects are becoming a significant economic and social phenomenon. Thousands exist today, with the number of developers participating in each ranging from a few to many thousands. The number of users of the software produced by open source software development projects ranges from few to many millions. Well-known examples of open source software having many users are the GNU/Linux computer operating system, Apache server software, and the Perl programming language.

To set a context for exploring the interest that the open source software phenomenon can hold for organization science researchers, we begin by briefly explaining the history and nature of open source software itself (the product). Next we outline key characteristics of the open source software development projects typically used to create and maintain such software (the development process).

Open Source Software

In the early days of computer programming commercial "packaged" software was a rarity—if you wanted a particular program for a particular purpose, you typically wrote the code yourself or hired it done. Much of the software development in the 1960s and 1970s was carried out in academic and corporate laboratories by scientists and engineers. These individuals found it a normal part of their research culture to freely give and exchange software they had written, to modify and build upon each other's software both individually and collaboratively, and to freely give out their modifications in turn. This communal behavior became a central feature of "hacker culture." (In communities of open source programmers, "hacker" is a very positive term that is applied to very talented and dedicated programmers.)[1]

In 1969 the U.S. Defense Advanced Research Project Agency (ARPA) established the ARPANET, the first transcontinental, high-speed computer network. This network eventually grew to link hundreds of universities, defense contractors, and research laboratories. Later succeeded by the Internet, it also allowed hackers to

exchange software code and other information widely, easily, and cheaply—and also enabled them to spread hacker norms of behavior.

The communal hacker culture was very strongly present among a group of programmers—software hackers—housed at the MIT's Artificial Intelligence Laboratory in the 1960s and 1970s (Levy 1984). In the 1980s this group received a major jolt when MIT licensed some of the code created by its hacker employees to a commercial firm. This firm, in accordance with normal commercial practice, then promptly restricted access to the "source code" of that software, and so prevented noncompany personnel—including MIT hackers who had participated in developing it—from continuing to use it as a platform for further learning and development. (Source code is a sequence of instructions to be executed by a computer to accomplish a program's purpose. Programmers write computer software in the form of source code, and also "document" that source code with brief written explanations of the purpose and design of each section of their program. To convert a program into a form that can actually operate a computer, source code is translated into machine code using a software tool called a compiler. The compiling process removes program documentation and creates a "binary" version of the program—a sequence of computer instructions consisting only of strings of ones and zeros. Binary code is very difficult for programmers to read and interpret. Therefore, programmers or firms that wish to prevent others from understanding and modifying their code will release only binary versions of the software. In contrast, programmers or firms that wish to enable others to understand and update and modify their software will provide them with its source code (Moerke 2000, Simon 1996).)

Richard Stallman, a brilliant programmer at MIT's Artificial Intelligence Laboratory, was especially distressed and offended by this loss of access to communally developed source code and also by a general trend in the software world towards development of proprietary software packages and the release of software in forms that could not be studied or modified by others. Stallman viewed these practices as morally wrong impingements upon the rights of software users to freely learn and create. In response, he founded the Free Software Foundation in 1985, and set out to develop and diffuse a legal mechanism that could preserve free access for all to the software developed by software hackers. His pioneering idea was to use the existing mechanism of copyright law to this end. Software authors interested in preserving the status of their software as "free" software could use their own copyright to grant licenses on terms that would guarantee a number of rights to all future users. They could do this by simply affixing a standard license to their software conveying these rights. The basic license developed by Stallman to implement this idea was the General Public License or GPL (sometimes referred to as "copyleft"—a play on the word "copyright"). Basic rights transferred to those possessing a copy of free software include the right to use it at no cost and the right to study its "source code," to modify it, and to distribute modified or unmodified versions to others at no cost. Licenses conveying similar rights were developed by others, and a number of such licenses are currently used in the open source field.

The free software idea did not immediately become mainstream, and industry was especially suspicious of it. In 1998, Bruce Perens and Eric Raymond agreed that a significant part of the problem resided in Stallman's term "free" software, which might understandably have an ominous ring to the ears of business people. Accordingly they, along with other prominent hackers, founded the "open source" software movement (Perens 1998). "Open source" software incorporates essentially the same licensing practices as those pioneered by the free software movement. It differs from that movement primarily on philosophical grounds, preferring to emphasize the practical benefits of such licensing practices over issues regarding the moral rightness and importance of granting users the freedoms offered by both free and open source software. The term "open source" is now generally used by scholars to refer to free or open source software, and that is the term we will use in the remainder of this paper.

Recently, open source software has emerged as a major cultural and economic phenomenon. Today, the number of open source software projects is rapidly growing, with a single major infrastructure provider and repository for open source software projects, Sourceforge.net, listing in excess of 50,000 projects and more than 500,000 registered users. A significant amount of software developed by commercial firms is being released under open source licenses as well. Current contributors of code to open source software projects are actively concerned with protecting user rights to freely use and improve and learn from open source computer code (O'Mahony 2002).

Open Source Software Development Projects

Software can be termed open source independent of how or by whom it has been developed: The term denotes only the type of license under which it is made available. However, the fact that open source software is freely accessible to all has created some typical open

source software development practices that differ greatly from commercial software development models—and that look very much like the "hacker culture" behaviors described earlier.

Because commercial software vendors typically wish to sell the code they develop, they sharply restrict access to the source code of their software products to firm employees and contractors. The consequence of this restriction is that only insiders have the information required to modify and improve that proprietary code further (see Meyer and Lopez 1995, also Young et al. 1996, Conner and Prahalad 1996). In sharp contrast, all are offered free access to the source code of open source software. This means that anyone with the proper programming skills and motivations can use and modify any open source software written by anyone. In early hacker days, this freedom to learn and use and modify software was exercised by informal sharing and code-velopment of code—often by the physical sharing and exchange of computer tapes and disks upon which the code was recorded. In current Internet days, rapid technological advances in computer hardware and software and networking technologies have made it much easier to create and sustain a communal development style at ever-larger scales. Also, implementing new projects is becoming progressively easier as effective project design becomes better understood, and as prepackaged infrastructural support for such projects becomes available on the Web.

Today, an open source software development project is typically initiated by an individual or a small group with an idea for something interesting they themselves want for an intellectual or personal or business reason. Raymond (1999a, p. 32) suggests: "Every good work of software starts by scratching a developer's personal itch . . . too often software developers spend their days grinding away for pay at programs they neither need nor love. But not in the (open source) world . . . " The project initiators also generally become the project "owners" or "maintainers" who take on responsibility for project management.[2] Early on, this individual or group generally develops a first, rough version of the code that outlines the functionality envisioned. The source code for this initial version is then made freely available to all via downloading from an Internet website established by the project. The project founders also set up mailing lists for the project that those interested in using or further developing the code can use to seek help, provide information, or provide new open source code for others to discuss and test. In the case of projects that are successful in attracting interest, others do download and use and "play with" the code—and some of these do go on to create new and modified code. Most then post what they have done on the project website for use and critique by any who are interested. New and modified code that is deemed to be of sufficient quality and of general interest by the project maintainers is then added to the "authorized" version of the code. In many projects the privilege of adding to the authorized code is restricted to only a few trusted "developers." These few then serve as "gate keepers" for code written by contributors who do not have such access (von Krogh et al. 2003).

Most who download open source software are free riders—only a relatively small proportion contribute to a project by developing code or in other ways. Open source projects do not pay participants for their services, and the motivations and characteristics of contributors vary. Most are strongly motivated by a personal or business use for the code that they develop—they are code "users" in our terminology. Other major sources of motivation include intrinsic rewards such as personal learning and enjoyment from programming. Most contributors are experienced, professional programmers. Some act as independent individuals, others are employees of organizations that support their participation (Lakhani and Wolf 2001).

Two brief case histories will help to further convey the flavor of open source software development projects.

Apache Server Software. Apache server software is used on Web server computers that host Web pages and provide appropriate content as requested by Internet browsers. Such computers are the backbone of the Internet-based World Wide Web infrastructure.

The server software that evolved into Apache was developed by University of Illinois undergraduate Rob McCool for, and while working at, the National Center for Supercomputing Applications (NCSA). The source code as developed and periodically modified by McCool was posted on the Web so that users at other sites could download, use, and modify and further develop it. When McCool departed NCSA in mid-1994, a small group of webmasters who had adopted his server software for their own sites decided to take on the task of continued development. A core group of eight users gathered all documentation and bug fixes and issued a consolidated patch. This *patchy* Web server software evolved over time into Apache. Extensive user feedback and modification yielded Apache 1.0, released on December 1, 1995.

In the space of four years and after many modifications and improvements contributed by many users, Apache has become the most popular Web server software on the Internet, garnering many industry awards for excellence. Despite strong competition from commercial

software developers such as Microsoft and Netscape, it is currently used by some 60% of the millions of websites worldwide. Modification and updating of Apache by users and others continues, with the release of new versions being coordinated by a central group of about 30 volunteers.

Fetchmail—An Internet E-mail Utility Program. Fetchmail is an Internet e-mail utility program that "fetches" your e-mail from central servers to your local computer. The open source project to develop, maintain, and improve this program was led by Eric Raymond (Raymond 1999a).

Raymond first began to puzzle about the e-mail delivery problem because he was personally dissatisfied with then existing solutions. "What I wanted was for my mail to be delivered on snark, my home system, so that I would be notified when it arrived and could handle it using all my local tools" (ibid., p. 31).

Raymond decided to try to develop a better solution. He began by searching databases in the open source world for an existing, well-coded utility that he could use as a development base. He knew it would be efficient to build upon others' related work if possible, and in the world of open source this practice is understood and valued. Raymond explored several candidate open source programs, and settled upon one in small-scale use called "popclient." He developed a number of improvements to the program and proposed them to the then maintainer of popclient. It turned out that this individual had lost interest in working further on the program, and so his response to Raymond's suggestions was to offer his role to Raymond so that he could himself evolve the popclient further as he chose.

Raymond accepted the role of popclient maintainer, and over the next months he improved the program significantly in conjunction with advice and suggestions from other users. He carefully cultivated his more active "beta list" of popclient users by regularly communicating with them via messages posted on a public electronic bulletin board set up for that purpose. Many responded by volunteering information on bugs they had found and perhaps fixed, and by offering improvements they had developed for their own use. The quality of these suggestions was often high because "...contributions are received not from a random sample, but from people who are interested enough to use the software, learn about how it works, attempt to find solutions to the problems they encounter, and actually produce an apparently reasonable fix. Anyone who passes all these filters is highly likely to have something useful to contribute" (ibid., p. 42).

Eventually, Raymond had arrived at an innovative design that he knew worked well because he and his beta list of codevelopers used it, tested it, and improved it every day. Popclient (now renamed fetchmail) became standard software used by millions of servers on the Internet. Raymond continues to lead the group of volunteers that maintains and improves the software as new user needs and conditions dictate.

We propose that the practices of open source software developers and communities will be of interest to researchers working in organization science for two major reasons. First, open source software projects present a novel and successful alternative to conventional innovation models. This alternative presents interesting puzzles for and challenges to prevailing views regarding how innovations "should" be developed, and how organizations "should" form and operate. Second, open source software development projects offer opportunities for an unprecedented clear look into their detailed inner workings. By the very nature of the way these projects operate, detailed and time-stamped logs of most interactions among community members and of project outputs are automatically generated. These logs are publicly available and open to the inspection of any researcher without special permission. This simple fact makes open source software development projects valuable as research sites for many types of studies.

In §2, following, we explore the general nature of the research opportunities that open source offers to those interested in organization science in general, and innovation models in particular. In §3 we offer a discussion regarding open research questions and the types of data available for open source software projects in order to ease access for researchers who are unfamiliar with these, and we offer some advice on the conduct of empirical studies on this phenomenon.

2. Open Source Software Projects—Exemplar of a "Private-Collective" Innovation Model

Society has a vital interest in encouraging and rewarding innovation. Presently, there are two major models characterizing how this may be done. The first, the "private investment" model, assumes that innovation will be supported by private investment and that private returns can be appropriated from such investments (Demsetz 1967). To encourage private investment in innovation, society grants innovators some limited rights to the innovations they generate via intellectual property law mechanisms such as patents, copyrights, and trade secrets. These rights, in turn, assist innovators in getting private returns from their innovation-related investments (Arrow 1962, Liebeskind 1996, Dam 1995).

In the private investment model, any free revealing or uncompensated "spillover" of proprietary knowledge developed by private investment will reduce the innovator's profits from its investment. It is therefore assumed that innovators will avoid such spillovers to the extent possible—although they will occur nonetheless (Audretsch and Feldman 1996, Audretsch and Stephan 1999, Harhoff et al. 2000). At the same time, the monopoly control that society grants to innovators under the private incentive model and the private profits they reap represent a loss to society relative to the free and unfettered use by all of the knowledge that the innovators have created. Society elects to suffer this social loss in order to increase innovators' incentives to invest in the creation of new knowledge.

The second major model for inducing innovation is termed the collective action model. This model applies to the provision of public goods, where a public good is defined by its nonexcludability and nonrivalry: If any user consumes it, it cannot be feasibly withheld from other users (Olson 1967, p. 14). The collective action model operates in science and elsewhere (e.g., Merton 1973, Aldrich 1999, Monge et al. 1998, McCaffrey et al. 1995, Coleman 1973, Eyerman and Jamison 1991, Hess 1998, Melucci 1999). It requires that contributors relinquish control of knowledge they have developed for a project and make it a public good by unconditionally supplying it to a "common pool." This requirement enables collective action projects to avoid the social loss problem associated with the restricted access to knowledge of the private investment model. At the same time, it creates problems with respect to motivating potential contributors to collective action projects.

Because contributions to a collective action project are a public good, potential beneficiaries of that good have the option of waiting for others to contribute and then free-riding on what they have done (Olson 1967). One solution to this problem is to supply some form of monetary or reputation or other subsidy to contributors to collective action projects to raise their level of motivation. For example, many societies provide monetary subsidies for basic research for this reason. The social structure of science itself then operates via norms of reciprocity and knowledge sharing among scientists to insure contributions to public goods are made, and to offer reputation-based rewards for good performance (Merton 1973, Stephan 1996).

In the case of open source software development projects, we see an interesting compound of the private investment and collective action models of innovation. We term this compound the "private-collective" innovation model. In this model, participants in open source software projects use their own resources to privately invest in creating novel software code. In principle, these innovators could then claim proprietary rights over their code, but instead they choose to freely reveal it as a public good. Clearly, the net result of this behavior appears to offer society the best of both worlds—new knowledge is created by private funding and then offered freely to all. However, it also creates an intriguing puzzle. As Lerner and Tirole (2000) put it: "Why should thousands of top-notch programmers contribute freely to the provision of a public good?"

Much of the research needed to answer that puzzle is yet to be done. Answering it will involve, we think, revisiting and easing some of the basic assumptions and constraints conventionally applied to the private investment and collective action models of innovation. In essence, we think that each of the two basic models—in an effort to offer "clean" and simple models for research—have excluded from consideration a very rich and fertile middle ground where incentives for private investment and collective action can coexist, and where a "private-collective" innovation model can flourish. We think this middle ground is where open source software projects in fact reside. The end result of exploring it will be, we think, a deeper understanding of a promising new mode of organization for innovation that can indeed deliver "the best of both worlds" to society under many conditions. In the remainder of this section we consider how open source software development practice deviates from the conventional assumptions of the private investment and collective action models of innovation. Then, we show how the conditions actually faced by open source software projects offer the basis for a novel, private-collective model for the motivation of innovation.

Open Source Software Project Deviations from the Private Investment Model of Innovation

Open source software development practice involves two major deviations from the private investment model of innovation as it is conventionally viewed. First, software users rather than software manufacturers are the typical innovators in open source. Second, open source innovators freely reveal the proprietary software that they have developed at their private expense.

With respect to the first deviation, recall that the private investment model of innovation is premised upon the idea that individuals or organizations will step forward and invest in the development of innovations if and as they expect such action to "pay" in terms of private rewards. The model places no additional con-

straint on who will tend to innovate. However, manufacturers rather than product users have traditionally been considered the most logical private developers of innovative products and services because private financial incentives to innovate seem to be higher for them than for individual or corporate users. After all, a manufacturer has the opportunity to sell what it develops to an entire marketplace of users while spreading development costs over a large number of units sold. Individual user-innovators, on the other hand, can typically expect to benefit financially only from their own internal use of their innovations. Benefiting from diffusion of an innovation to the other users in a marketplace would require some form of intellectual property protection followed by licensing. Both are costly to attempt, with very uncertain outcomes (Taylor and Silberston 1973, Cohen et al. 2000, Levin et al. 1987, Kogut and Metiu 2001).

In the case of open source software projects one observes that, contrary to conventional expectation, the bulk of contributions—and therefore the bulk of private innovation-related investment—are made by developers that are *users* of that software, either as individual users or user firms, rather than by software manufacturers. Why should this be? The most fundamental reason is that software users can profit by using open source software or open source software improvements that they develop. In contrast, there is no commercial market for open source software—because its developers make it freely available as a public good. This eliminates manufacturers' direct path to appropriating returns from private investment in developing open source products, and so often eliminates their incentive to innovate. (Note, however, that manufacturers may find indirect paths to profiting from open source software projects and so may contribute to them. For example, IBM may profit from developing improvements to the open source program GNU/Linux if these improvements enhance Linux's functioning with a complementary good—proprietary computer software or hardware—that IBM does sell.) Of course, to say that manufacturers have a disadvantage with respect to reaping private rewards from open source software innovations is not the same as saying that users have sufficient private incentives to innovate based upon internal use only. However, in a number of fields it has been shown that such incentives can indeed suffice, and that users often do innovate (von Hippel 1988).

Next we come to the second major deviation that open source software development practice displays relative to the private investment model of innovation. Users in open source communities typically freely reveal their innovations by, for example, posting improvements and code on project websites where anyone can view and download them for free. As was noted earlier, free revealing does not make sense from the point of view of the private investment model of innovation. After all, as the conventional reasoning goes, innovating users under budget constraints spend money and time to create their innovations and revealing their developments without compensation to noninnovating users, either directly or via a manufacturer, should represent a loss of potential private returns that users should strive to avoid.

How are we to understand such behavior? From the viewpoint of the private investment model of innovation, users should only freely reveal their innovations when the costs of free revealing are less than the benefits. It has been argued that such conditions can hold in many fields, including open source software (Harhoff et al. 2003). First, it is pointed out that a number of phenomena, ranging from network effects to increased sales of complementary goods, can actually increase innovators' private benefits if and as free revealing causes their innovations to be diffused more widely. Second, it is pointed out that, independent of any potential private gain from free revealing of an innovation, any private losses associated with this action will typically be quite low. In brief overview, this line of argument begins by noting that there are two kinds of costs associated with revealing an innovation: those associated with the loss of proprietary rights to intellectual property, and the cost of diffusion. With respect to the former, innovators can expect low losses from free revealing if they have low rivalry with potential adopters of their innovations and/or expect gains from the increased diffusion of their innovation that free revealing will cause. In the case of open source software projects, contributors are diverse, and it is highly likely that at least some potential contributors of a given innovation will see themselves as having low rivalry with respect to potential adopters. (Indeed, many contributors to open source software projects are students who do not have any basis for commercial rivalries with other potential adopters; see Lakhani and Wolf 2001, Hermann et al. 2000.) And when *some* holders of a given innovation face low rivalry conditions and so are likely to freely reveal, it does not benefit any holder of that same information to keep it private.

In the case of open source software projects, the costs that an innovator incurs to freely reveal the novel code he or she has developed *and* widely diffuse it also are low. Open source software project participants simply post that code on the appropriate project Internet site. The act of posting and the act of retrieving the posted information by others are both nearly costless. When the expected costs associated with free revealing are low, even a low level of reward can be sufficient to induce the

behavior. As Lerner and Tirole (2000) and von Krogh (2002) observe, adequate rewards can be provided to participants in open source software projects in a variety of forms, including elevated reputations, expected reciprocity, and incentives to help build a community.

Open Source Software Project Deviations from the Collective Action Model of Innovation

The collective action model of innovation is a response to market failure. The model can be applied to the creation of public goods ranging from provision of a public bridge to provision of open source software. As was mentioned earlier, a public good is defined by its nonexcludability and nonrivalry: If any user consumes it, it cannot feasibly be withheld from other users (Olson 1967). Nonexcludability leads directly to the central dilemma for the collective action model: If users who do *not* contribute to a public good—"free riders"—can benefit from that good on equal terms with those who do contribute, how can one motivate users to contribute rather than free ride? The collective action literature has responded to this central dilemma by placing a great deal of emphasis on the importance of recruiting and properly motivating participants in a successful collective action project in order to increase the attractiveness of contributing relative to free riding.

With respect to successfully recruiting contributors to a collective action task, conventional theory predicts that both the specification of project goals and the nature of recruiting efforts should matter a great deal (McPhail and Miller 1973, Snow et al. 1980, Snow and Benford 1992, Benford 1993). Thus, researchers have pointed out that direct and stable social relationships between recruiters and potential participants are important, so that recruiters will have more knowledge of individual motivations and be more effective in defining a rewarding goal (Oliver and Marwell 1988, Taylor and Singleton 1993). The nature of effective recruiting strategies has also been explored (Taylor and Singleton 1993, Benford 1993, Snow and Benford 1992).

With respect to the free-rider problem, Axelrod (1984) notes that it has been found that the reward for cooperation can be higher than the reward for defection in a multiround prisoner's dilemma game with no fixed endpoint. Therefore, he suggests that it would be effective to convince participants in a collective action project that they are engaged in long-term cooperative relationships. Schwartz and Paul (1992) argue that it can be effective to convince potential contributors that the importance of "group fate" outweighs the cost incurred of contributing.

It has also been argued that the creation and deployment of selective incentives for contributors are essential to the success of collective action projects (e.g., Friedman and McAdam 1992, Oliver 1980). Thus, projects may elect to use specific social categories and to bestow credentials on their members, based on observed efforts and/or skills. Individuals may then gain private benefits from such credentials in the form of enhanced social relations, enhanced reputation, privileged access to social relations, and so on.[3] The importance of selective incentives to successful collective action in turn suggests that small groups will be most successful at executing collective action projects because selective incentives can then be carefully tailored for each group member (Olson 1967, pp. 43–52; Fireman and Gamson 1979; Taylor and Singleton 1993). Also, monitoring of individual efforts is easier in a small group where people meet and communicate face to face (Ostrom 1998). In particular, if knowledge of group members overlaps with respect to the task, an appraisal of others' efforts can be made that is consistent with the appraisal of one's own efforts (Osterloh and Frey 2000).

Interestingly, successful open source software projects do not appear to follow any of the guidelines for successful collective action projects just described. With respect to project recruitment, goal statements provided by successful open source software projects vary from technical and narrow to ideological and broad—and from precise to vague and emergent (for examples, see goal statements posted by projects hosted on Sourceforge.net).[4] Further, such projects typically engage in *no* active recruiting beyond simply posting their intended goals and access address on a general public website customarily used for this purpose (for examples, see the website named "Freshmeat"). Also, projects have shown by example that they can be successful even if large groups—perhaps thousands—of contributors are involved. Finally, open source software projects seem to expend no effort to encourage contributing over free riding. Anyone is free to download code or seek help from project websites, and no apparent form of moral pressure is applied to make a compensating contribution (e.g., "If you benefit from this code, please also contribute . . . ").

What can explain these deviations from expected practice? We propose that the *private* rewards to those who contribute to open source software collective action projects are considerably stronger than those available to free riders. This in turn enables the deviations from expected collective action practice that are observable in open source software projects. We develop this idea in our discussion of the private-collective innovation model that follows next.

Open Source Software Projects as an Illustration of a Private-Collective Model of Innovation

The central deviation we believe that open source software projects display with respect to the assumptions about incentives embedded in the private investment and the collective action models of innovation is that contributions to open source software development are not pure public goods—they have significant private elements even after the contribution has been freely revealed. More specifically, the private-collective model of innovation occupies the middle ground between private investment and collective action models by:

• Eliminating the assumption in private investment models that free revealing of innovations developed with private funds will represent a loss of private profit for the innovator and so will not be engaged in voluntarily. Instead it proposes that under common conditions free revealing of proprietary innovations may not involve a loss of profit to innovators who developed those innovations with private funds. Indeed, under some conditions free revealing may actually result in a net gain in private profit for the innovator. For example, free revealing can increase innovation diffusion and so increase an innovator's innovation-related profits through network effects.

• Eliminating the assumption in collective action models that a free rider will be able to obtain benefits from the completed public good that are equal to those a contributor can obtain. Instead, it proposes that contributors to a public good can *inherently* obtain private benefits that are tied to the development of that good. These benefits are available only to project contributors and not to free riders and represent a form of "selective incentives" for project participation that need not be managed by collective action project personnel.

To explore these ideas, consider first that contributors to an open source software project must engage in problem solving to create novel code. When they freely reveal this code to the project, it becomes a public good. However, the problem-solving process and effort used to produce the code have other important outputs as well, such as learning and enjoyment, and a sense of "ownership" and control over their work product. In open source and other software projects the technical learning opportunities can be enormous (e.g, Kohanski 1998, Himanen 2001, Hermann et al. 2000). Previous coding and learning in turn might increase the user's returns on learning in future activity (Arthur 1997). Surveys of individuals who contribute to open source software projects find that on average they regard personal learning and enjoyment derived from programming to be very important motivators for their project participation (Lakhani and Wolf 2001, Hermann et al. 2000).

Contributors to open source software projects also report valuing the sense of ownership of and control over their work product that they experience in open source software projects and do not experience in programming work they carry out for hire (Lakhani and von Hippel 2000, Moon and Sproull 2000, Hermann et al. 2000). This difference makes sense. For-profit programming firms often seek to reduce development costs and control quality by closely monitoring what programmers do and how they do it (e.g., Cusumano 1992, Sawyer and Guinan 1998). In contrast, contributors to open source software projects choose the project, the task they will work on, and their technical approach to that task to suit their own interests. Outputs from the code-writing process such as learning and enjoyment are *private* benefits that are available to contributors but *not* to free riders. Interestingly, some of the private benefits, such as the private learning garnered from critiques and corrections supplied by others, may *only* be obtainable if the code itself is revealed to others.

Consider next that software code is information and so is a nonrival good. This means that any number of people can use the good simultaneously: My use of the software does not interfere with your use of it. It also means that I as a developer can contribute my code as a public good and at the same time use it for my private and perhaps somewhat different purposes. In the case of open source software, modules are regularly created by individual users or user firms for private purposes and are tailored to their individual needs (Pavlicek 2000). Then, they are openly revealed and contributed to the project as a public good for whatever general use there may be. To the extent that the conditions faced by the contributor differ from those faced by free riders, the contributor may be in a more favorable position than free riders to gain private benefit from the code he contributes.

Contributors to open source software projects may also get private benefits from participating in the project "community" that are not available to free riders (Raymond 1999a, Wayner 2000, Lerner and Tirole 2000, Moon and Sproull 2000). If the cooperation among users is intense and sustainable, these might even outweigh individual user rewards (von Krogh 1998, 2002). As John Elster remarks (1986, p. 132): "Cooperation reflects a transformation of individual psychology so as to include the feeling of solidarity, altruism, fairness, and the like. Collective action ceases to become a prisoner's dilemma because members cease to regard participation as costly: It becomes a benefit in itself, over and above the public good it is intended to produce." Recent developments in economic theory support Elster's conjecture.

Thus, Rabin (1993) and Fehr and Schmidt (1999) have shown that a game that in material payoffs constitutes a prisoner's dilemma can be transformed into a coordination game in which cooperation is also an equilibrium outcome if pecuniary motivations *and* social motivations are taken into account.

Recall now the puzzle posed by Lerner and Tirole (2000) noted earlier in the paper: "Why should thousands of top-notch programmers contribute freely to the provision of a public good?" If we combine our observations regarding the interlinked public and private aspects of contributions made to open source software projects with our earlier observation regarding the generally low competitive advantage associated with keeping privately generated code private, then we have a reasonable and reasonably likely answer to this question: Programmers contribute freely to the provision of a public good because they garner private benefits from doing so.

In sum, the emerging phenomenon of open source software development obviously does not undermine either the private investment or the collective action models of innovation. However, it does make clear the utility of combining both into a "private-collective" incentive model that can more effectively address the interlinked private and collective incentive structures observable in that field, and perhaps elsewhere as well.

3. Discussion

Early in this paper we pointed out that open source software projects appeared to be potentially interesting and important to researchers interested in organization science, for two reasons. First, such projects appeared to offer "the best of both worlds"—a happy combination of the private investment and collective action models of innovation, and second, open source software projects offer opportunities for an unprecedented clear look into their detailed inner workings because detailed and time-stamped logs of most interactions among community members and of project outputs are automatically generated. In this final section of our paper, we discuss both of these points in more detail.

Some Research Questions

We have seen that participants in open source software projects commonly use their private budget to create innovations they freely reveal as a public good. We have also seen those contributors retain and gain significant private benefits from this action: They retain private benefits from their work process such as learning and enjoyment, and they gain benefits associated with community participation as well. This suggests a rational economic basis for a "private-collective" innovation incentive. Studies of an integrated private-collective innovation model could greatly increase our understanding of how public goods are and may be collectively provided.

Recall that in the collective action literature, the most important solution to the free-rider problem is found in the creation and awarding of selective incentives (Friedman and McAdam 1992, Oliver 1980). The important selective incentives and private benefits associated with participation in projects, such as private functional value obtained from code that is contributed, and the learning and enjoyment associated with code writing, are those that the contributor applies and "awards" to him or herself. This may largely obviate the need for collective action project managers to either provide or monitor selective incentives within the confines of a project. It would be important to explore the nature of the private and community incentives acting upon open source contributors more deeply. In the course of this research it will be important to distinguish between incentives impacting firms that may assign employees to contribute to open source software projects, and incentives impacting individual contributors.

Self-provision of private rewards for contributions to a public good have very significant implications for the governance of collective action projects. A major argument for central governance of collective action projects springs from the need to discourage free riding. Thus conventional reasoning suggests that in the larger group, collective action becomes increasingly fragile (e.g., Raub 1988), since social relationships become increasingly scattered and ephemeral, and interests increasingly diverse. When the group increases in size, the impact of any individual's participation in producing the collective good is negligible, and a self-interested, rational individual will choose to free-ride under these conditions (Hardin 1971). The cost of an individual's decision to free-ride is spread over a greater number of people, and the cost of organizing and using selective incentives to induce cooperation of individuals increases as well (Marwell and Oliver 1993). It becomes increasingly costly for each group member to monitor and sanction others' free-riding behavior. Eventually, as the group grows, the monitoring costs outgrow the costs of contributing, and jointly they outweigh the rewards from the public good itself. The expected outcome is that the public good will be underprovided.

To sustain collective action in large groups, conventional reasoning suggests that it will be rational for project members to turn to or to install a central authority or leader who specializes in monitoring of group

members and enforcement of sanctions against free riding (Hardin 1982; see also Swanson 1992, Stroup 2000). The central authority moderates the cost of using the selective incentive, but under this authoritarian regime, collective action only succeeds if the regulatory interests that mandate cooperation overcome those changes in interests that would encourage defection. In other words, leaders can formulate norms and create incentives that safeguard compliance with norms (e.g., normative leadership) and maintain the value of social categories through the control of credentialing processes.

The "self-rewarding" of important private benefits by participants in open source software projects that are not also available to free riders may considerably diminish participants' concern about free riders. Indeed, informal observation in the field of open source software development suggests that contributors actually regard free riders as an asset. Free riders that adopt open source code without contributing to it nonetheless increase the "market share" and importance of the project and may help set de facto standards. Also, some who free-ride with respect to code creation and other major project tasks are not free riders with respect to all project-related tasks. For example, they may report software bugs that they encounter to project contributors. The more users and program usage there is, the faster bugs get identified and the faster contributors can fix them with benefit to all program users (Raymond 1999a).

The immediate nature of private rewards associated with the development of code may also mean that active recruiting of members is not significant to project success. Of course, it may also mean that contributors to an open source software project may have smaller incentives to stay with a project long term than they would if rewards were consciously allocated by project managers or community members in a gradual manner. We need to understand how and why project members join and leave particular projects, and the nature and emergence of social categories in such projects.

It would also be important to explore the nature of social integration of individuals in their communities. Users whose identity is known to the community enjoy greater benefits from revealing their innovations than do those who are less integrated (see also Wenger 1998, 2000; Taylor 1989; Calhoun 1988). This is so because their ideas, bug reports, viewpoints, or code can be reviewed and commented upon by other users, and in terms of learning benefits the group's feedback can be direct and specific. Open source entrepreneurs therefore have an incentive to integrate important users socially, by such means as listing the important developers in a project (von Krogh et al. 2003). Social integration might not prevent withdrawal from the project through punishment, but perhaps through the individual utility derived from a social category such as a core-developer status. Hence, open source software entrepreneurs might seek to make this category valuable, rare, and membership restricted. If users' motives change, the learning rewards have been exhausted, and/or the value of the social category depreciates, users are likely to reduce the level of their participation. So what is the nature of social integration in the private-collective innovation model, and how important is this for users' involvement with a particular project?

Finally, the question of "active integration" also raises questions regarding the role of leadership and central authority. Many observers of the open source software phenomenon point to the paramount role many leaders have had in the development of an open source software project (Pavlicek 2000). In fact, it has already been argued that various forms of leadership extending beyond simple authoritarian leadership can have a positive effect on motivation of contributors for collective action (see Coloner 1995, Frolich et al. 1971, Salisbury 1969). However, leadership is likely to be very different from the leadership we have observed in the private investment—and perhaps even the collective action model of innovation. Experimental evidence shows that groups can perform worse in producing collective action when outside agencies monitor group activities and appropriate resources for this activity, primarily because groups find it difficult to agree on rules for monitoring and sanctioning to be bestowed on this authority (Schmitt et al. 2000). Interestingly, one of the norms clearly expressed in the hacker community is that work cannot be mandated and enforced by a leader (Himanen 2001, Raymond 1999a), nor does the leader have any monetary incentives or legal basis to induce or enforce cooperation. So, what is the importance of leadership for sustaining activity in distributed innovation, how do leaders emerge, and what are the various functions they perform? If it is true that leaders who can choose who is a member of a social category secure a more talented group, and hence more effective production of public goods (Schwart and Tomz 1997), leadership in distributed innovation might in fact be analogous to that performed by a playing coach. These issues need much more exploration in future studies.

A deeper understanding of a private-collective model of innovation incentives within open source software projects will help us to understand how far such a model is applicable to other fields as well. We think that it is likely that it can apply quite broadly. Consider that the simultaneous existence of public and private aspects

of "public goods production" is possible in the case of physical products as well as information products. For example, an architect contributing services to the design of a public bridge may gain private learning and enjoyment, as well as reputation advantages from his labor, that will not be equally available to free riders. Also, one who contributes large amounts of resources to the creation of a public good can sometimes gain a higher level of influence on the project's goal and directions that can be used to advance private agendas (McCarthy and Zald 1977, Fireman and Gamson 1979). Note also that even physical products consist of knowledge and information at the design stage—one can freely reveal CAD models of an airplane design as readily as one can freely reveal software code.

Free revealing of privately developed innovations as public goods is also not unique to open source software projects. It also has been observed in areas ranging from industrial equipment innovations and closed source software developed by firms to sporting equipment innovations developed by individual end consumers (Allen 1983, Morrison et al. 2000, Harhoff et al. 2000, Franke and Shah 2001). Also, the existence of user innovation communities in the instance of sporting equipment innovations has been documented (Franke and Shah 2001).

What may be unique to knowledge and information products is that in these fields we see users carrying out the entire innovation process for themselves—no manufacturer required. Thus, open source software projects encompass the entire innovation process, from design to distribution to field support and product improvement. Such "full-function" user innovation and *production* communities are possible only when self-manufacture and/or distribution of innovative products directly by users can compete with commercial production and distribution. In the case of open source software this is possible because innovations can be "produced" and distributed essentially for free on the Web, software being information rather than a physical product. In the case of innovations embodied in physical products, one would expect that while users would innovate, general diffusion would require the involvement of manufacturers. This is because physical products must be produced and physically distributed and these activities involve significant economies of scale (von Hippel 2001).

Some Starting Points for Open Source Software Innovation Research

Open source software research projects have some access points and technical aspects that may be unfamiliar to those just beginning research in this field. Most projects are hosted on a few major sites like Sourceforge.net. Researchers considering a first empirical research project on open source software development will find it useful to begin by browsing Sourceforge.net to become familiar with standard project infrastructures such as code listings and mailing lists and logs devoted to different specialized functions.

Much of the activity transpiring in open source software projects is a matter of public record. Code written for the project and accepted into the "official" version is available online. In active projects, this code is modified and added to often—perhaps several times per day. Records of what is "committed" to the official code, and by whom, is recorded in a publicly accessible Concurrent Versioning System (CVS) log. CVS is an important software tool used by many open source projects. Its function is to keep track of changes made to the source code by project developers. It also stores the project's source code along with programmers' written comments that explain their work in detail. In many projects the privilege of adding authorized code to the CVS is restricted to only a few trusted developers. These few then serve as gate keepers for code written by contributors who do not have such access (von Krogh et al. 2003).

Interactions among project members are generally posted in the form of messages on public Internet sites maintained by projects. One example of such a site is Geocrawler. These interactions are all time-stamped and organized into "threads" consisting of an initial message and all others that directly respond to it. It is important to note that a simple thread search, say under a heading of "functionality X," might not uncover all discussions about this particular part of the software architecture. Relevant discussions about this functionality might also appear in different threads. Furthermore, some projects also organize their discussions in separate domains—for example, one reserved for technical development and the other for general interest and comments on the public use of the software.

A description of and empirical study illustrating the use of CVS log data is Koch and Schneider (2000), who were interested in the range of efforts contributors put into a project over time. They accessed publicly available data on the GNU Network Object Model Environment (GNOME) project in the CVS repository and in discussion groups. They provided descriptive statistics (e.g., contributor profiles, lines of code added or deleted, number of postings to the mailing list) and cluster analyses to identify contributor groups and program files, applying lines of code as the discriminating variable. This research used established metrics of software development productivity and successfully captured the

dynamic nature of GNOME. However, it can be very valuable to verify metrics and variables with project experts in early empirical studies of a new arena. For example, lines of code reflect quantitative aspects of the project's final source code, but that metric fails to capture the importance of the code to the overall project architecture.[5]

An empirical study that describes and illustrates the analysis of discussion thread data is Yamauchi, Yokozawa, Shinohara, and Ishida (2000). They focused on the Free Berkeley Software Distribution (FreeBSD) Newconfig., a project aimed at modifying the FreeBSD software for devices that use PCMCIA cards, and the GNU (Gnu Not Unix) GCC project, which aims at developing an improved compiler for the GNU system, including the GNU/Linux variant. Their interest was in how an open source software project could achieve smooth coordination, agreement on design, and innovation using limited media. They used a mixture of methods tracking threads on public discussions, analyzing the contents of messages and task descriptions to identify communication patterns among users. They proposed "rational" social norms governing the communication: Users make their behavior logically plausible to the community, base decisions on technical performance criteria, and have a bias for action/programming rather than project planning.

In general, summary data such as number of messages by number of participants are easy to develop by straightforward analysis of publicly available logs (indeed, some summary statistical data is available on open source software project host sites like Sourceforge.net). However, researchers should be aware that much user communication happens beyond public email. Thus, Internet Relay Chat (real time chatting on the Internet), private e-mail, or direct communication between users can have significant value for studies of motives, incentives, community development, coordination, and technical decision making in projects (Lakhani et al. 2001).

Interpretation of subtle matters relevant to organization researchers will be aided by having a good contextual and behavioral understanding of project activities, and a broad set of data and methods might then be valuable. For example, Lee and Cole (2000) opted for an inductive approach (Glaser and Strauss 1967) to analyze data related to the development of the GNU/Linux "kernel"—the portion of the GNU/Linux operating system that coordinates the functioning of its various components. Their data sources covered secondhand interviews, the Linux kernel mailing list during the years 1995–2000, and archival data of the Linux source code (1,9 Mio. LOC).

Finally, it can be very useful to create an "intellectual genealogy" for an open source development project at an early stage in an empirical research project. Such a genealogy traces major changes over time in the software architecture and identifies decisions and code contributions that have had a major impact on the evolution of the software, software functionality, and project. With such a genealogy in hand, researchers will be able to distinguish critical from noncritical project attributes and behaviors, and so be in a better position to deepen our understanding of issues central to the effective functioning of open source software development projects.

4. Conclusion

There are clearly many interesting puzzles in open source software research projects that can trigger the interest of organization scholars for years to come. Answering some of them might even lead to substantial rethinking of the very concept of "organization for innovation" and to a better understanding of innovation among distributed users who derive utility from freely revealing their information-based innovation to produce a collective good. We hope that we have stimulated the interests of some readers, and look forward to joining with them in further explorations of this very interesting topic!

Acknowledgments

The authors are grateful for insightful comments from Kaye Schoonhoven, Sebastian Spaeth, Petra Kugler, John Seely Brown, and Simon Gaechter.

Endnotes

[1]hacker n. [originally, someone who makes furniture with an axe] 1. A person who enjoys exploring the details of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn only the minimum necessary. 2. One who programs enthusiastically (even obsessively) or who enjoys programming rather than just theorizing about programming. 3. A person capable of appreciating hack value. 4. A person who is good at programming quickly. . . 8. [deprecated] A malicious meddler who tries to discover sensitive information by poking around. Hence "password hacker," "network hacker." The correct term for this sense is cracker. (Jargon File 2001). See also Halbert (1997).

[2]"The owner(s) [or 'maintainers'] of an open source software project are those who have the exclusive right, recognized by the community at large, to *redistribute modified versions.*" . . . "According to standard open-source licenses, all parties are equal in the evolutionary game. But in practice there is a very well-recognized distinction between 'official' patches [changes to the software], approved and integrated into the evolving software by the publicly recognized maintainers, and 'rogue' patches by third parties. Rogue patches are unusual and generally not trusted" (Raymond 1999a, p. 89).

[3]The nature of the incentives to be deployed marks a difference between classical organization theory and collective action theory: "Organization theory is most developed for organizations based on material incentives, whereas most (collective action) bind people with solidarity and purposive (immaterial) incentives" (Zald 1992, p. 336). An immaterial selective incentive would result from members creating and enforcing social categories (Tajfel and Turner 1979, Tajfel 1982): "People's sense of who they are in terms of some meaningful social category (e.g., occupational, gender, status, age) that distinguishes how they interact with those inside from those outside the category" (Roy and Parker-Gwin 1999, p. 206).

[4]As a specific example of a project with an emergent goal, consider the beginnings of what became the Linux open source software project. In 1991, Linus Torvalds, a student in Finland, wanted a Unix operating system that could be run on his PC equipped with a 386 processor. Minix was the only software available at that time but it was commercial, closed source, and it traded at USD 150.-. Torvalds found this too expensive, and started development of a Posix-compatible operating system, later known as Linux. Torvalds did not immediately publicize a very broad and ambitious goal, nor did he actively attempt to recruit contributors. He simply expressed his private motivation in a message he posted on July 3, 1991, to the USENET newsgroup comp.os.minix (Wayner 2000, p. 55) as follows: *Hello netlanders, Due to a project I'm working on (in minix), I'm interested in the posix standard definition.* (Posix is a standard for UNIX designers. A software using POSIX is compatible with other UNIX-based software.) *Could somebody please point me to a (preferably) machine-readable format of the latest posix-rules? Ftp-sites would be nice.* In response, Torvalds got several return messages with Posix rules and people expressing a general interest in the project. By early 1992, several skilled programmers contributed to Linux and the number of users increased by the day. Today, Linux is the largest open source development project extant in terms of number of developers, and in the server software market it is second only to Microsoft in terms of servers that use it.

[5]The number of lines of code is not a good measure of a project's qualitative progress, and hence is often disputed as a metric to reflect a project's output (Koch and Schneider 2000). As illustration of the uncertain link between quality and quantity, consider that hackers typically favor neat and compact programs that achieve a given function using minimum lines of code. This is evidenced by the very popular 5K-competition, where people can submit programs up to 5 kilobytes, and where the program with the most impressive functions under these constraints wins an award.

References

Aldrich, H. 1999. *Organizations Evolving*. Sage, London, U.K.

Allen, Robert C. 1983. Collective invention. *J. Econom. Behavior and Organ.* **4**(1) 1–24.

Arrow, K. 1962. Economic welfare and the allocation of resources for inventions. R. R. Nelson, ed. *The Rate and Direction of Inventive Activity.* Princeton University Press, Princeton, NJ, 609–625.

Arthur, W. B. 1997. Path-dependence, self-reinforcement, and human learning. W. B. Arthur, ed. *Increasing Returns and Path Dependence in the Economy*. The University of Michigan Press, Ann Arbor, MI, 133–158.

Audretsch, D. B., M. P. Feldman. 1996. R&D spillovers and the geography of innovation and production. *Amer. Econom. Rev.* **86**(3) 630–640.

——, P. E. Stephan. 1999. Knowledge spillovers in biotechnology: Sources and incentives. *J. Evolutionary Econom.* **9** 97–107.

Austin, R. D. 2001. The effects of time pressure on quality in software development: An agency model. *Inform. Systems Res.* **12**(2) 195–207.

Axelrod, R. 1984. *The Evolution of Cooperation*. Basic Books, New York.

Benford, R. D. 1993. You could be the hundreth monkey: Collective action frames and vocabularies of motives within the nuclear disarmament movement. *Sociological Quart.* **34** 195–216.

Calhoun, C. 1988. The radicalism of tradition: Community strength or venerable disguise and borrowed language? *Amer. J. Sociology* **88**(5) 886–924.

Cohen, W. M., R. R. Nelson, J. P. Walsh. 2000. Protecting their intellectual assets: Appropriability conditions and why U.S. manufacturing firms patent (or not). Working Paper 7522, National Bureau of Economic Research.

Coleman, J. 1973. *The Mathematics of Collective Action*. Aldine, Chicago, IL.

Coloner, J. M. 1995. Leadership games in collective action. *Rationality and Society* **7**(2) 225–247.

Conner, K. R., C. K. Prahalad. 1996. A resource-based theory of the firm: Knowledge versus opportunism. *Organ. Sci.* **7**(5) 477–501.

Cusumano, M. C. 1992. Shifting economies: From craft production to flexible systems and software factories. *Res. Policy* **21**(5) 453–480.

Dam, K. W. 1995. Some economic considerations in the intellectual property protection of software. *J. Legal Stud.* **24**(2) 321–377.

Demsetz, H. 1967. Towards a theory of property rights. *Amer. Econom. Rev.* **57** 347–359.

Elster, J. 1986. *An Introduction to Karl Marx.* Cambridge University Press, Cambridge, U.K.

Eyerman, R., A. Jamison. 1991. *Social Movements: A Cognitive Approach*. Penn State Press, University Park, PA.

Fehr, E., K. M. Schmidt. 1999. A theory of fairness, competition, and cooperation. *Quart. J. Econom.* **114**(3) 817–868.

Fireman, B., W. H. Gamson. 1979. Utilitarian logic in the resource mobilization perspective. M. N. Zald, J. D. McCarthy, eds. *The Dynamics of Social Movements*. Winthrop, Cambridge, MA.

Franke, Nik, Sonali Shah. 2001. How communities support innovative activities: An exploration of assistance and sharing among innovative users of sports equipment. Working paper, MIT Sloan School of Management, Cambridge, MA.

Friedman, D., D. McAdam. 1992. Collective identity and activism: Networks, choices, and the life of a social movement. A. D. Morris, C. McClurg, eds. *Frontiers in Social Movement Theory*. Yale University Press, New Haven, CT, 156–173.

Frolich, N., J. A. Oppenheimer, O. Young. 1971. *Political Leadership and Collective Goods*. Princeton University Press, Princeton, NJ.

Glaser, B., A. Strauss. 1967. *The Discovery of Grounded Theory*. Weidenfeld and Nicholson, London, U.K.

Halbert, D. 1997. Discourses of danger and the computer hacker. *Inform. Soc.* **13**(4) 361–374.

Hardin, R. R. 1971. Collective action as an aggregate N-prisoner's dilemma. *Behavioral Sci.* **16** 472–481.

——. 1982. *Collective Action*. Johns Hopkins Press, Baltimore, MD.

Harhoff, D., J. Henkel, E. von Hippel. 2000. Profiting from voluntary information spillovers: How users benefit from freely revealing their innovations. Working paper, MIT Sloan School of Management, Cambridge, MA.

Hermann, S., G. Hertel, S. Niedner. 2000. *Linux Study Homepage*, www.psychologie.uni-kiel.de/linux-study/writeup.html.

Hess, S. 1998. Individual behavior and collective action towards the environment: An economic framework based on the social customs approach. *Rationality and Soc.* **10**(2) 203–222.

Himanen, P., Linus Torvalds, Manuel Castells. 2001. *The Hacker Ethic*. Random House, New York.

Jargon File, version 4.3.1. 2001. www.tuxedo.org/~esr/jargon/html /entry/hacker.html, June 29. [The Jargon File is a collective online work by computer hackers. In print form, it is Raymond, Eric. 1999b.]

Koch S., G. Schneider. 2000. Results from software engineering research into open source development projects using public data. Work paper, University of Economics and Business Administration, Vienna, Austria.

Kogut, B., A. Metiu. 2001. Open source software development and distributed innovation. *Oxford Rev. Econom. Policy* **17**(2) 248–264.

Kohanski, D. 1998. *Moths in the Machine*. St. Martin's Griffin, New York.

Lakhani, K., E. von Hippel. 2000. How open source software works: "Free" user-to-user assistance. Working Paper 4117, MIT Sloan School of Management, Cambridge, MA.

——, R. Wolf. 2001. Does free software mean free labor? Characteristics of participants in open source communities. Boston Consulting Group Survey Report, Boston, MA. (Available at www.osdn.com/bcg/.)

Lee, G. K., R. Cole. 2000. The Linux kernel development as a model of open source knowledge creation. Working paper, University of California, Berkeley, Berkeley, CA.

Lerner, J., J. Tirole. 2000. The simple economics of open source. NBER Working paper series, WP 7600, Harvard University, Cambridge, MA.

Levin, R. C., A. K. Klevorick, R. R. Nelson, S. G. Winter. 1987. Appropriating the returns from industrial research and development. *Brookings Papers on Econom. Activity* **3** 783–831.

Levy, Steven. 1984. *Hackers*. Anchor/Doubleday, New York.

Liebeskind, J. P. 1996. Knowledge, strategy, and the theory of the firm. *Strategic Management J.* **17** 93–107.

Marwell, G., P. Oliver. 1993. *The Critical Mass in Collective Action: A Micro-Social Theory*. Cambridge University Press, Cambridge, U.K.

McCaffrey, D. P., S. R. Faerman, D. W. Hart. 1995. The appeal and difficulties of participative systems. *Organ. Sci.* **6**(6) 603–627.

McCarthy, J. D., M. N. Zald. 1977. Resource mobilization and social movements: A partial theory. *Amer. J. Sociology* **82** 1212–1241.

McPhail, C., D. Miller. 1973. The assembling process: A theoretical and empirical examination. *Amer. Sociological Rev.* **38** 721–735.

Melucci, A. 1999. *Challenging Codes: Collective Action in the Information Age*. Cambridge University Press, Cambridge, U.K.

Metiu, A., B. Kogut. 2002. Distributed knowledge and the global organization of software development. opensource.mit.edu.

Merton, R. K. 1973. The sociology of science: Theoretical and empirical investigations. N. W. Storer, ed. University of Chicago Press, Chicago, IL.

Meyer, M. H., L. Lopez. 1995. Technology strategy in a software products company. *J. Product Innovation Management* **12**(4) 194–306.

Moerke, K. A. 2000. Free speech to a machine. *Minnesota Law Rev.* **84**(4) 1007–1008.

Monge, P. R., J. Fulk, M. E. Kalman, A. J. Flanagin, C. Parnassa, S. Rumsey. 1998. Production of collective action in alliance-based interorganizational communication and information systems. *Organ. Sci.* **9**(3) 411–433.

Moon, J. Y., L. Sproull. 2000. Essence of distributed work: The case of the Linux kernel. *First Monday* **5**(11)

Morrison, P. D., J. H. Roberts, E. von Hippel. 2000. Determinants of user innovation and innovation sharing in a local market. *Management Sci.* **46**(12) 1513–1527.

Oliver, P. E. 1980. Rewards and punishment as selective incentives for collective action: Theoretical investigations. *Amer. J. Sociology* **85** 1356–1375.

——, G. Marwell. 1988. The paradox of group size in collective action: A Theory of the Critical Mass II. *Amer. Sociological Rev.* **53**(1) 1–18.

Olson, M. 1967. *The Logic of Collective Action*. Harvard University Press, Cambridge, MA.

O'Mahony, S. 2002. Guarding the commons: How open source contributors protect their work. Working paper, Stanford University, Stanford, CA.

Osterloh, M., B. Frey. 2000. Motivation, knowledge transfer, and organizational forms. *Organ. Sci.* **11**(5) 538–550.

Ostrom, E. 1998. A behavioral approach to the rational choice theory of collective action. *Amer. Political Sci. Rev.* **92**(1) 1–22.

Pavlicek, R. C. 2000. *Embracing Insanity: Open Source Software Development*. Sams, Indianapolis, IN.

Perens, Bruce. 1998. The open source definition. http://perens.com/Articles/OSD.html.

Rabin, M. 1993. Incorporating fairness into game-theory and economics. *Amer. Econom. Rev.* **83**(5) 1281–1302.

Raub, W. 1988. Problematic social situations and the large number dilemma: A game theoretical analysis. *J. Math. Sociology* **13** 311–357.

Raymond, E. 1999a. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly, Sebastopol, CA.

——. 1999b. *The Hacker's New Dictionary*. MIT Press, Cambridge, MA.

Roy, W. G., R. Parker-Gwin. 1999. How many logics of collective action? *Theory and Soc.* **28** 203–237.

Salisbury, R. H. 1969. An exchange theory of interest groups. *Midwest J. Political Sci.* **13** 1–32.

Sawyer, S., P. J. Guinan. 1998. Software development: Process and performance. *IBM Systems J.* **37**(4) 552–569.

Schmitt, P., K. Swope, J. Walker. 2000. Collective action with incomplete commitment: Experimental evidence. *Southern Econom. J.* **66** 829–855.

Schwart, E. P., M. R. Tomz. 1997. The long-run advantages of centralization for collective action. *Amer. Political Sci. Rev.* **92**(3) 685–693.

Schwartz, M., S. Paul. 1992. Resource mobilization versus the mobilization of people. A. D. Morris, C. McClurg, eds. *Frontiers in Social Movement Theory*. Yale University Press, New Haven, CT, 205–223.

Simon, E. 1996. Innovation and intellectual property protection: The software industry perspective. *Columbia J. World Bus.* **31**(1) 30–37.

Snow, D. A., R. D. Benford. 1992. Master-frames and cycles of protest. A. D. Morris, C. McClurg, eds. *Frontiers in Social Movement Theory*. Yale University Press, New Haven, CT, 133–154.

——, L. Zurcher, S. Ekland-Olson. 1980. Social networks and social movements: A microstructural approach to differential recruitment. *Amer. Sociological Rev.* **45** 787–801.

Stephan, P. 1996. The economics of science. *J. Econom. Literature* **XXXIV** 1199–1235.

Stroup, R. L. 2000. Free-riders and collective action revisited. *Independent Rev.* **4**(4) 485–501.

Swanson, G. E. 1992. Doing this together—Some basic forms of agency and structure in collective action and some explanations. *Soc. Psych. Quart.* **55**(2) 94–117.

Tajfel, H. 1982. *Social Identity and Integroup Relations*. Cambridge University Press, Cambridge, U.K.

——, J. C. Turner. 1979. An integrative theory of inter-group conflict. W. G. Austin, S. Worchel, eds. *The Social Psychology of Intergroup Relations*. Brooks/Coole, Monterey, CA.

Taylor, C. T., Z. A. Silberston. 1973. *The Economic Impact of the Patent System: A Study of the British Experience*. Cambridge University Press, Cambridge, U.K.

Taylor, M. 1989. Rationality and revolutionary collective action. M. Taylor, ed. *Rationality and Revolution*. Cambridge University Press, Cambridge, U.K., 63–97.

——, S. Singleton. 1993. The communal resource: Transaction costs and the solution of collective action problems. *Politics and Society* 195–215.

von Hippel, E. 1988. *The Sources of Innovation*. Oxford University Press, New York.

——. 1998. Economics of product development by users: The impact of sticky local information. *Management Sci.* **44**(5) 629–644.

——. 2001. Innovation by user communities: Learning from open source software. *Sloan Management Rev.* (July)

von Krogh, G. 1998. Care in knowledge creation. *California Management Rev.* **40**(3) 133–153.

——. 2002. The communal resource and information systems. *J. Strategic Inform. Systems* **11**(2) 85–107.

——, S. Spaeth, K. Lakhari. 2003. Community, joining and specialization in open source software innovation: A case study. *Res. Policy*. Forthcoming.

Wayner, P. 2000. *Free for All*. Harper Business, New York.

Wenger, E. 1998. *Communities of Practice*. Cambridge University Press, Cambridge, U.K.

——. 2000. Strategic communities and knowledge diffusion. *Sloan Management Rev.* **41**(3) 9.

Yamauchi, Y., Yokozawa, M., Shinohara, T., T. Ishida. 2000. *Collaboration with Lean Media: How Open-Source Software Succeeds.* ACM. CSCW '00, Philadelphia, PA.

Young, G., K. G. Smith, C. M. Grimm. 1996. Austrian and industrial organization perspectives on firm level competitive activity and performance. *Organ. Sci.* **7**(3) 243–254.

Zald, M. 1992. Looking backward to look forward: Reflections on the past and future of the resource mobilization research program. A. D. Morris, C. McClurg, eds. *Frontiers in Social Movement Theory*. Yale University Press, New Haven, CT, 326–350.