ELSEVIER

Research Policy 32 (2003) 1149–1157

research policy

www.elsevier.com/locate/econbase

Editorial

Special issue on open source software development

Georg von Krogh [a,*], Eric von Hippel [b,1]

[a] *Institute of Management, University of St. Gallen, Dufourstrasse 48, St. Gallen CH-9010, Switzerland*
[b] *Sloan School of Management, MIT, 50 Memorial Drive, Cambridge, MA 02139, USA*

Abstract

This special issue of Research Policy is dedicated to new research on the phenomenon of open source software development. Open Source, because of its novel modes of operation and robust functioning in the marketplace, poses novel and fundamental questions for researchers in many fields, ranging from the economics of innovation to the principles by which productive work can best be organized. In this introduction to the special issue, we provide a general history and description of open source software and open source software development processes, plus an overview of the articles.

Keywords: Open source software development; Editorial

1. Introduction

Open source software development projects are generally Internet-based networks or communities of software developers. The software they develop is made freely available to all that adhere to the licensing terms specified by the open source project. Open source software projects and development processes have spread rapidly and widely, and many thousands exist today. The number of developers participating in each project ranges from a few to many thousands, and the number of users of the software produced by open source software development projects range 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.

The phenomenon of open source software is of great scholarly interest to many. Its robust functioning in the marketplace together with its novel modes of operation pose major and exciting new questions regarding fundamental matters ranging from the economics of innovation to the principles by which productive work can best be organized. In recognition of the interest and importance of this topic, *Research Policy* offers this special issue as a forum for some exciting new work and voices on open source software. In this brief introduction to the special issue, we first provide a quick overview of the history and functioning of open source and free software. Next we link each of the papers published in this special issue to a general topology of questions of interest to many in this field.

* Corresponding author. Tel.: +41-71-224-23-63; fax: +41-71-224-23-55.
E-mail addresses: georg.vonkrogh@unisg.ch (G. von Krogh), evhippel@mit.edu (E. von Hippel).
[1] Tel.: +1-617-253-7155; fax: +1-617-253-2660.

0048-7333/03/$ – see front matter
doi:10.1016/S0048-7333(03)00054-4

2. Open source software: history and characteristics[2]

A good way to get a first grasp of open source software is to observe how, throughout its history, it has differed from commercial software. Today, in the software industry, firms and individuals appropriate and secure financial rewards from their "packaged" software products by two principle means. First, the software industry uses licensing arrangements based on copyright law. A software license provides individuals and groups with the legal rights to use a piece of software often in return for a licensing fee (e.g. Dam, 1995; Granstrand, 1999). In current commercial practice, most software is licensed rather than sold to a third party as intellectual property. A software license typically restricts the numbers of computers the software can run on, the number of software users, backups, and the simultaneous use of backups.

Second, many commercial software firms protect the software's "source code", which 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).

In the early days of computer programming such commercial, protected, and "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.[3])

In 1969 the US Defense Advanced Research Project Agency (DARPA) 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 non-company personnel—including MIT hackers who had participated in developing it—from continuing to use it as a platform for further learning and development.

[2] In this section, we draw upon three excellent histories of the open source software movement: "The Cathedral and the Bazaar" written by Raymond (1999), "Hackers" written by Levy (1984), and a more recent account "Rebel Code" written by Moody (2001).

[3] 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, 2002).

Richard Stallman, a brilliant programmer at the 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 protected under licenses that prevented it from being 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 about to develop and diffuse a legal mechanism that could preserve free access for all to the software developed by software hackers (Moody, 2001). His pioneering idea was to use the existing mechanism of copyright law to this end. Software authors interested in preserving the status of their work 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 that conveyed these rights.

The basic license developed by Stallman to implement this idea was the General Public License (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, the right to study its "source code," to modify it, and to distribute modified or unmodified versions to others at no cost. Others developed licenses conveying similar rights, and currently a number of such licenses are 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 use in this article.

2.1. 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. Rather, these practices look very much like the "hacker culture" behaviors described earlier.

Because commercial software vendors typically wish to license or sell the code they develop, they have an incentive to 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 to the firm" possess the information required to modify and improve that proprietary code further (see Meyer and Lopez, 1995, for more on commercial software development). In sharp contrast, all are offered free access to the source code of open source software. This means that anyone with the proper programming knowledge and motivations can use, study, and modify any open source software written by anyone. In early hacker days described earlier, this freedom to learn and use and modify software was exercised by informal sharing and co-development of code—often by the physical sharing and exchange of computer tapes, disks, or punch cards 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 designs becomes better understood, and as infrastructures for such the management of projects become 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 (1999, 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.[4] Early on, this individual or group may develop an initial, rough version of the code that outlines the functionality envisioned. The source code for this first 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. Those interested in using or further developing the code can consult this list in order to get help running the software or writing code. The list can also be used to provide information or distribute new open source code for others to discuss and test further.

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 based on their own interests. Most then post what they have done on the project website for use and critique by anyone who is interested. New and modified code that is deemed to be of sufficient quality and of general appeal 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" who then become part of a "community" of developers. Most contributors are experienced, professional programmers. Some act as independent individuals volunteering to develop code, others are employees of organizations that support their participation (e.g. Lakhani et al., 2002).

[4] "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, 1999, p. 89).

3. Contents of the special issue

In the call for papers, the editors encouraged contributions from scholars working in diverse fields, and placed few restrictions on research methods used. Our hope was to offer a collection of papers on open source software that would both address interesting questions and illustrate a number of pathways to doing interesting work in this field. Contributors responded to our call, and the articles in this special issue do reflect a good variety of interesting questions and methods and disciplines. In what follows, we briefly describe the content of each article in this special issue under three major headings: (1) motivations of contributors to open source software projects; (2) how the innovation process functions; (3) competitive dynamics of open source software. Finally, we describe a research note that makes some interdisciplinary linkages with theory and research in social anthropology. Table 1 provides an overview of some contributions the various articles make to these three topics.

3.1. Motivations of contributors

A central puzzle raised by the success of open source software development is why thousands of top-notch programmers appear to be contributing freely to the provision of a public good. A number of researchers have begun to explore this important question. Emerging findings are easing the puzzle by showing that contributing to open source software is in fact a "private-collective" activity, involving important elements that remain private to code creators even as the code itself becomes a public good (von Hippel and von Krogh, 2003). Important among these are *process*-related elements associated with the task of coding such as learning and enjoyment (Lakhani et al., 2002). Also, the enhancement of private reputations can play a role (Lerner and Tirole, 2000). In this special issue, three empirical articles address aspects of this important matter.

The first article, by Hertel, Niedner, and Herrmann is entitled "Motivation of software developers in open source projects: an Internet-based survey of contributors to the Linux Kernel." These authors test two extant models in the social science literature that appear to fit the circumstances of individual contributors to open source software projects. One, by

Table 1
Contributions in overview

Topics	Issues	Contributions						
		Hertel, Niedner, and Herrmann	O'Mahony	Franke and von Hippel	von Krogh, Spaeth, and Lakhani	Bonaccorsi and Rossi	West	Zeitlyn
Motivations	Individual's motivations	User's motives to improve code and tolerance of time investments		Increased user satisfaction with innovation tools				
	Protection of software under open source license		Practices to allow open source software to be governable and publically available					
Innovation process	Community-related mechanisms				Social scripts for joining a community			
	Evolution of software architecture				The evolution of the software architecture relates to the specialization of newcomers in a project			
Competitive dynamics	Rivalry between commercial and open source software					Diffusion of open source software in a market dominated by commercial software	The impact of open source software on commercial software manufacturers	
	Cooperation between manufacturers and the open source software movement						Hybrid strategies for computer platforms that include open source software	
Interdisciplinary linkage	A view on open source software from social anthropology							Gift giving and kinship amity

Klandermans (1997) is an explanation of incentives to participate in social movements. A second deals with motivational processes in small work teams, particularly "virtual teams" with members working in different places and coordinating their work mainly via electronic media (Hertel, 2002; Hertel et al., 2002). The authors find good fit between both models and data they derived from a survey of 141 contributors to the Linux kernel. They demonstrate that contributors' involvement with the Linux kernel project is influenced by their identification as Linux developers—an interesting finding with respect to the importance of community. They also find contributors to be motivated by pragmatic motives to improve their own software and also group-related factors such as their perceived indispensability for the team with which they were working.

Siobhan O'Mahony's article is an ethnographic study entitled "Guarding the commons: how community managed software projects protect their work." O'Mahony proposes an important addition to the considerations generally raised with respect to motives behind public goods production. She points out that contributors to open source software do not simply provide open access to their work without concern for its ultimate fate. Instead, they have a very active concern that their work remain part of the "commons," and energetically protect their work to this end. Based on interviews with more than 70 contributors to open source projects, she explores how open source project members encourage compliance with the terms of their projects' open source licenses in various ways. For example, they may exercise sanctioning via on-line discussions, and may make use of trademarked brands and logos in order to insure that the intellectual property they have contributed remains in the commons.

Franke and von Hippel contribute an article entitled "Satisfying heterogeneous user needs via innovation toolkits: the case of Apache security software." This paper explores a frequently cited reason for contributing to open source software: people innovate to better satisfy their own needs. In an empirical study of the security needs of users of Apache security software, they show that user needs are highly heterogeneous, and that therefore many Apache users are dissatisfied with standard Apache security functionality. Because Apache is open source software, it offers a solution to the customization problem that is not available in the case of "closed" products users are free to modify the software to better fit their individual needs. The authors find that users that do modify the software are more satisfied as a result. The authors conclude by suggesting that this same strategy could be emulated in many other fields by offering users "toolkits for user innovation" to enable them to easily make product changes and improvements without manufacturer assistance.

3.2. Innovation process

A second major issue explored by authors in this special issue concerns how the development process works. In commercial software practice, the development of software products hinges on tight management of the processes: software firms regulate the relationship to their programmers through carefully drafted employment contracts, divide labor and allocate work responsibilities along the development process, and take precautions to prevent employees from leaking software-related trade secrets and information to competitors. For-profit programming firms 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; Austin, 2001). Open source software development differs fundamentally from these practices. Von Krogh, Spaeth, and Lakhani contribute an article entitled "Community, joining script, and specialization: a case study", that explores a novel aspect of open software development practices: how contributors join projects.

Based on a clinical study of Freenet, an open source project for peer-to-peer computing, the authors show that the software architecture and functionality are governed by a community consisting of developers who can commit code to the authorized version of the software. When joining such a community contributors adhere to a social "joining script": they put in considerably more technical and sophisticated work than the average contributor to Freenet. After being admitted into the community, the new developers specialize in mundane software development tasks before moving on to more complex and technically difficult work. The authors also find that newcomers often give early gifts in terms software modules and features that enhance the software architecture and functionality.

This gift giving is rewarding to the newcomer: he can contribute software to the project by continuing to work on those modules or features that he knows best.

3.3. Competitive dynamics

A third general topic addressed by two contributions to this special issue is the competitive dynamics introduced by open source software. First, what competitive dynamics may unfold from *rivalry* between commercial software and their open source variant? Andrea Bonaccorsi and Cristina Rossi, in their article "Why open source software can succeed" propose that open source software can shed new light on an old problem in innovation theory: how new technologies can compete in an environment dominated by technological standards. Diffusion of technologies in the presence of network externalities can explain why open source software is becoming widespread in an environment previously dominated by established proprietary standards. The authors note that recent developments in the theory of critical mass in the diffusion of technologies can be helpful to understand these phenomena. They develop a simulation model in a SWARM environment in order to identify and evaluate the relevant factors in the diffusion of open source software. A number of interesting findings emerge from this exercise. On the one hand the diffusion of new technologies is associated with positive beliefs among users. In the presence of well-established software standards effective diffusion of open source software hinges on several sources of network externality. On the other hand, in the absence of incumbent advantages for established commercial software, and given a belief among users towards open source software, the only way commercial software can control the market is for the software manufacturer to engage in fierce competition on quality and understake massive investments in R&D. A robust general result of the simulation is that under many plausible conditions open source software and commercial software are likely to coexist, even in the limit.

Second, some manufacturers choose to circumvent rivalry with the open source software movement and search for ways to *cooperate* with it. This raises a general issue of what strategies manufacturers pursue and what rewards ensue from these. In the article entitled "How open is open enough? Melding proprietary and open source platform strategies" Joel West argues that, since the early 1960s, a succession of computer platforms have provided an integrated architecture of dominant hardware and software. These platforms provided the basis for building complimentary assets such as application software and peripherals, and manufacturers' strategies for building and leveraging these platforms have provided them with economic rent. Like Profs. Bonaccorsi and Rossi, Prof. West is concerned with the conditions under which open source software can successfully compete with dominant commercial technologies, but adds that manufacturers' involvement in open source software matters too. Based on the cases of IBM, Sun Microsystems, and Apple Computer, he shows significant differences among manufacturers' resource commitments to open source and their hybrid strategies of melding open source and proprietary platforms. The author also proposes under what conditions a manufacturer may prefer a mix of strategies to the pure-open or pure-closed alternatives.

3.4. An interdisciplinary linkage

In our view, at this early stage research and theory building on open source software development should be receptive to debates in several fields and attempt to make linkages with important work done in other disciplines. The last article in this special issue is a research note written from a social anthropological perspective. In "Gift economies in the development of open source software: anthropological reflections", David Zeitlyn's intention is to show how anthropological theory may help future research. He offers some important ideas regarding gift giving behavior in open source software development, by applying the concept of "kinship amity". He suggest that family-oriented structures can enhance our understanding of the contributions individuals make to open source software.

Acknowledgements

We follow the attractive tradition of open source software projects in publicly acknowledging the many contributions made to the development of this special issue. First, we would like to thank the authors who sent in manuscripts for review. Interestingly, many were by young academics who contributed excellent

papers. Next, we thank those who served as reviewers for the papers submitted to the special issue, many of whom reviewed more than one paper! Accustomed to labor in noble anonymity, they will be surprised to find themselves listed with thanks. Next, we thank the two senior *Research Policy* editors, Keith Pavit and Gary Pisano, who approved our proposal to create this special issue of *Research Policy* and supported us along the way. Our editorial assistant, Stefan Haefliger, deserves great thanks for the extremely valuable assistance he provided in coordinating the editorial process. We are also very grateful for the financial support from the University of St. Gallen. Our thanks also to Susan Lees from University of Sussex, as well as Edith Boomers and Lisa Muscolini from Elsevier Science, for their great support on publishing matters, and to all our colleagues at both the Sloan School and at University of St. Gallen, who supported this endeavor.

Reviewers for the special issue

Ritu Agarwal	University of Maryland
Rafael Andreu	IESE
Rob Austin	Harvard University
Jim Bessen	Independent
Paul Carlile	Massachusetts Institute of Technology
Michael Cusumano	Massachusetts Institute of Technology
Giovanni Dosi	Scuola Superiore S. Anna
Paul Duguid	University of California at Berkeley
Thomas Eberle	University of St. Gallen
Elgar Fleisch	University of St. Gallen
Simon Gächter	University of St. Gallen
Bob Galliers	London School of Economics and Political Science
Robert Grant	Georgetown University
Dietmar Harhoff	University of Munich
Sirkka L. Jarvenpaa	University of Texas at Austin
Justin Johnson	Cornell University
Stefan Koch	University of Vienna
Bruce Kogut	INSEAD
Richard N. Langlois	University of Connecticut
Kwanghui Lim	National University Singapore
Janne Ljungberg	Göteborg University
Bengt-Åke Lundvall	University of Aalborg
Alan D. MacCormack	Harvard University
Anca Metiu	INSEAD
Eric Monteiro	Norwegian University of Science and Technology
Jae Yun Moon	New York University
David Mowery	University of California at Berkeley
Julia Porter Liebeskind	University of Southern California
Emanuela Prandelli	Bocconi University
Eric Raymond	Independent
Ron Sanchez	IMD
Mohan Sawhney	Northwestern University
Lee Sproull	New York University
Jean Tirole	IDEI Toulouse
John Van Maanen	Massachusetts Institute of Technology
Steven Weber	University of California at Berkeley
Etienne Wenger	Independent
Ulrich Witt	University of Jena

References

Austin, R.D., 2001. The effects of time pressure on quality in software development: an agency model. Information Systems Research 12 (2), 195–207.

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

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

Granstrand, O., 1999. The Economics and Management of Intellectual Property. Edward Elgar, Cheltham.

Hertel, G., 2002. Management virtueller teams auf der basis sozialpsychologischer modelle. In: Witte, E.H. (Ed.), Sozialpsychologie Wirtschaflicher Prozesse. Pabst Publishers, Lengerich, Germany, pp. 172–202.

Hertel, G., Konradt, U., Orlikowski, B., 2002. Managing distance by interdependence: goal setting, task interdependence, and team-based rewards in virtual teams, submitted for publication.

Jargon File, 2002. The On-Line Hacker Jargon File, Version 4.3.3, 20 September 2002. http://www.tuxedo.org/~esr/jargon/html/index.html (Raymond, E., 1996. The Jargon File is a collective on-line work by computer hackers. The New Hacker's Dictionary, third ed. MIT Press, Cambridge, MA).

Klandermans, B., 1997. The Social Psychology of Protest. Basil Blackwell, Oxford.

Lakhani, K., Wolf, B., Bates, J., DiBona, C., 2002. The Boston Consulting Group Hacker Survey. Available at: http://www.osdn.com/bcg.

Lerner, J., Tirole, J., 2000. The Simple Economics of Open Source, NBER Working Paper Series, WP 7600. Harvard University, Cambridge, MA.

Levy, S., 1984. Hackers. Anchor/Doubleday, New York.

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

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

Moody, G., 2001. Rebel Code. Perseus Publishing, Cambridge, MA.

Perens, B., 1998. The Open Source Definition. Available at: http://perens.com/articles/osd.html.

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

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

Simon, E., 1996. Innovation and intellectual property protection: the software industry perspective. Columbia Journal of World Business 31 (1), 30–37.

Von Hippel, E., von Krogh, G., 2003. The private-collective innovation model in open source software development. Organization Science, in press.