ELSEVIER

European Economic Review 45 (2001) 819–826

EUROPEAN ECONOMIC REVIEW

www.elsevier.com/locate/econbase

The open source movement: Key research questions

Josh Lerner[a,b,*], Jean Tirole[c,d,e]

[a]*Harvard Business School, Morgan Hall, Room 395, Boston, MA 02163, USA*
[b]*National Bureau of Economic Research, Cambridge, MA, USA*
[c]*IDEI and GREMAQ (UMR 5603 CNRS), Toulouse, France*
[d]*CERAS (URA 2036 CNRS), Paris, France*
[e]*Massachusetts Institute of Technology, Cambridge, MA, USA*

Abstract

The paper analyzes the incentives of individual programmers and of commercial companies to participate in open source projects. While these incentives are in our opinion well accounted for by the economic paradigm, much empirical and theoretical work is still needed to answer the many interesting questions suggested by the open source movement.

JEL classification: L23; L31; L86

Keywords: Software; Technological innovation; Research and development

1. Introduction

Open source software has attracted substantial attention in recent years. The open source process has been heralded by some of its proponents, such as Raymond (1999), as a superior way of producing code. Whether open source software is indeed 'faster, better, cheaper' is still an object of controversy, but there is little doubt that open source projects have been tremendously successful in recent years.

* Corresponding author. Harvard Business School, Morgan Hall, Room 395, Boston, MA 02163, USA. Tel.: 617-495-6065; fax: 617-496-7357.

E-mail addresses: jlerner@hbs.edu (J. Lerner); tirole@cict.fr (J. Tirole).

0014-2921/01/$ - see front matter
PII: S0014-2921(01)00124-6

While much of the popular attention has been devoted to Linux, which is viewed as a major competitive threat to the Microsoft operating systems, a number of other Unix-based open source programs have made substantial inroads in their respective segments. For example, Apache, a free server program that runs on 56% of web servers,[1] is a favorite of web-masters, who appreciate its reliability, its ability to run on old machines, its price,[2] and its customizability (as programmers can most easily modify and extend the code). Sendmail underlies the routing of e-mails over the Internet. The Perl language is used for writing Common Gateway Interface scripts that are the standard way of delivering dynamic content on the Web. Such breakthroughs have been extensively covered in the media, and are even creating political ripples, from China's official endorsement of Linux to the US President's Information Technology Advisory Committee recommendation that the federal government back 'open source software as an alternative path for software development'.

The corporate response to the open source movement is equally interesting; new companies such as Red Hat and VA Linux have undergone well-publicized initial public offerings. Such companies do not directly make money from the open source programs (which, after all, are freely available), but rather charge for complementary services (documentation, installation software, utilities, etc.). Meanwhile, established commercial software companies have implemented 'open source strategies'. Many (e.g., Sybase, Oracle, Sun, IBM, Computer Associates) are porting programs to the Linux environment. For example, Intel has announced that its forthcoming 64-bit processor will run Linux, Corel has produced WordPerfect for Linux, and Apple is mixing open source software with proprietary Mac software to create a Unix server for Macintosh. Other software vendors have played a more proactive role by turning their code over to the open source community. For example, after losing ground to Microsoft's Internet Explorer, Netscape released in 1998 the source code of its Communicator 5.0 browser as an open source project named Mozilla. Commercial companies also often have a rather lenient attitude regarding their employees' involvement in open source projects.[3]

A natural initial question is what is open source software? Roughly, being open source requires that the source code, and not only the object code (the sequence of 1's and 0's that computers actually use), be made available to everyone, and that the modifications made by its users also be turned back to the community. The details vary with the license adopted for the program. Some

[1] This information is from Netcraft, Ltd. (http://www.netcraft.com/survey).

[2] It is free, although – as for other software programs – the relevant measure of price is the total cost of ownership (TCO), which includes the cost of implementation.

[3] To be sure, this subsidy to the open source movement is not always deliberate. In large, loosely run organizations, the line manager may be aware of the programmer's involvement, but the upper echelons may not. It is still the case that companies often do not actively discourage open source involvement.

of the key criteria included in the Open Source Definition[4] are (a) the royalty free redistribution of the program, (b) the release of the source code, and (c) the requirement that all modifications be distributed under the same terms as the license of the original software. By way of contrast, there are no restrictions on the software distributed along with (e.g., on the same medium as) the open source software. The Open Source Definition guidelines are otherwise flexible; for example, they accommodate restrictive licenses such as the General Public License. Open source software should not be confused with shareware (which is freely distributed, but whose source code remains proprietary) and public domain software (which is not licensed and is thus usable by everyone without constraint).

Commentators often feel that open source software challenges the economists' paradigm in several respects:

- *Individual incentives*: Why do top-notch programmers choose to write code that is released for free? Is this "gift economy" (Raymond, 1999) consistent with the self-interested-economic-agent paradigm?
- *Corporate strategies*: Why do commercial companies allocate some of their talented staff to open source programs? Why do software vendors initiate open source projects?
- *Organizational behavior*: Is the apparently anarchistic process of open source production, in which no one tells anybody else what to do, a new model of business organization?
- *Innovative process*: How does the new process fit with the conception of the innovative process driven by intellectual property rights (patents, copyrights and trade secrets) that we have inherited from Arrow (1962) and Schumpeter (1942)?

The paper makes no headway in the construction of a new theoretical paradigm. Rather, its more limited goals are (a) to argue that, contrary to appearances, the open source movement is relatively well accounted for by standard economic theory and (b) to point at areas in which further research would be desirable.

2. A challenge to received views on incentives and the innovation process?[5]

This section discusses incentives for programmers and software vendors to engage in open source projects.

[4] http://www.opensource.org/osd.html.

[5] This section builds on Lerner and Tirole (2000)

2.1. *Why do programmers participate*?

To explain why programmers freely contribute to the public good, economists (and indeed several open source leaders) are suspicious of the altruism hypothesis: the view that contributions are driven by pure generosity, or a sense of duty to give back to a community that has provided a useful piece of code. Of course, altruism exists and can do marvels in certain circumstances, but the altruistic hypothesis fails to explain why programmers do not focus their generosity on more needy beings and why free riding would be less pervasive than in biotechnology or other industries.

Part of the reason why people contribute is that, for some, the cost of contributing is not that high. First, they may have learned Unix (from which many open source programs are derived) in high school and college and thus find programming straightforward. Second, many contributors are sophisticated users who need to remove a bug or tailor the code to their specific applications. Having done so, they turn the code back to the community. They may then even see others improve on their modifications, increasing their private benefit further. This incentive corresponds to a well-known benefit of open source software: while external beta-testers of commercial software can point out problems in the program, users of open source software see the code itself and either are able to fix it or at least to identify where they think the problem is.[6]

While the private cost of participating in an open source project may be reasonable, the delayed benefit is non-negligible. Open source programmers' contributions (which are typically rather well defined and associated with the individual contributors) are well recognized. Project coordinators will list frequent contributors on the project's server and many observers (including those from commercial companies) will scrutinize contributions to the most popular projects. Frequent open source contributors furthermore have had ready access to venture capital: for example, former open source programmers started Sun and Netscape.[7] Besides these traditional career concerns incentives, programmers are also motivated by the (closely related) peer recognition motive. Like anybody else, programmers highly value being esteemed for their contributions.

To be certain, commercial software companies such as Microsoft try to emulate some of these incentives by giving developers the right to present their

[6] To be sure, software vendors do share their code with some privileged users; but they carefully select and negotiate detailed licensing agreements with such partners in order to protect their intellectual property rights. The sharing of commercial code is actually more often motivated by the desire to encourage complementary innovations than to have bugs fixed.

[7] A common misperception is that thousands or tens of thousands of programmers write popular open source software. The process, like the writing of commercial code, is actually quite elitist. For example, Mockus et al. (2000) report the top 15 developers have contributed from 83% to 91% of changes in the Apache code. By way of contrast, the reporting of bugs is much more democratic.

work to outside engineers, by promoting 'star' programmers to the rank of distinguished engineer, and by recognizing contributions with stock options. Yet, there is no doubt that the motivation provided by career concerns and ego gratification can be equally strong in an open source environment. Indeed, open source projects may prove attractive ports of entry into the market for higher positions or prestige for clever system administrations and students without particularly exciting prospects in their current positions.

2.2. *What constitutes a good project and a good leadership*?

Open source projects are often described by the media as anarchistic. True enough, nobody instructs others on what to do. Yet, successful open source projects are shepherded by well-respected leaders or leadership teams. The leadership posts on a developers' or core mailing list what it feels are the main problems (others being reported on secondary lists), selects the most appropriate solutions (a proposal may work but be undesirable as a general solution, not be portable to other platforms, etc.), and thereby shapes the recognized version of the software (the official release). While the official release by no means constrains parallel developments, the moral authority of the leadership tends to make the recognized solutions a focal point for the community.

Being a good leader requires more than talent in programming. As Max Weber would have predicted, the open source leadership must provide an initial vision, communicate clear procedures, and be perceived as fair, so contributors are not worried that decisions will be polluted by ego, commercial or political biases. The leadership also must not solve all challenging problems themselves and leave only the less glamorous tasks to others. (Open source programmers' career concern and ego gratification incentives require that they work on challenging problems.) And it must make sure that software design is modularized enough so as to permit a very decentralized, rather uncoordinated mode of production. In this respect, it is interesting to note that the Mozilla project encountered difficulties because the source code of the Communicator 5.0 browser was highly intertwined. (Much effort was devoted in 1999 to rewrite the code in a more modular way.)

Not all projects have good open source potential. The sophisticated-user bias created by the individual incentives discussed above means that programmers are less enthusiastic about developing for the mass audience such items as friendly graphical user interfaces or office suites[8] or about providing documentation. Rather, such tasks as the development of compilers and software for web servers have traditionally been emphasized.

[8] Exceptions include the KDE and GNOME open source projects, which are developing, respectively, the KOffice and StarOffice suite programs.

Many open source leaders now recognize the value of mixing commercial elements (complementary programs, documentation, and support) with open source programs. The lingering question though is that of mass-market applications. A substantial strength of Windows is that external application developers are provided several years in advance with a precise description of the future version's application programming interfaces (APIs) for use in programming. Furthermore, the developers know that some effort will be put toward maintaining backward compatibility in the future (although no commitments may be made). Challenges for the open source community also include avoiding the splintering of versions as a result of the product differentiation strategies of vendors such as Red Hat and VA Linux (who are likely to want to tailor their offerings) and insuring backward compatibility.[9]

2.3. Why do software vendors participate?

The commercial software companies' open source strategies are rather transparent. First, they try to make money on complementary services. Second, by letting some of their programmers get involved in open source, they keep abreast of open source developments, which allows them (a) to better know the competition, and (b) to develop an absorptive capacity both to be able to incorporate open source ideas into commercial software and to spot talented open source programmers for hiring purposes, and (c) to attract programmers who want to work in an intellectually challenging environment. Third, they may embrace an open source project to preempt the development of a standard around a technology owned by a powerful rival. We will discuss the release of proprietary code in the next section.

3. Some research questions

The open source movement suggests several interesting research questions, some of which transcend the software and information industry:

3.1. Opening proprietary code

Open source projects rarely start from scratch. Linux and the Apache web server are largely derived from software provided by academic or semi-academic (Bell Labs, Xerox Palo Alto Research Center) institutions. More and more of the original software is now being provided, however, by traditional for-profit companies.

[9] Another challenge will be the absence of liability in case clients with sensitive applications suffer substantial losses from a bug in the open source software (commercial vendors may be subject to such liability).

A key issue is the choice of governance mechanism for these projects. Microsoft, when it released ActiveX (which the company spent over $100 million developing), gave management responsibility to an independent group (the Open Group) in an attempt to commit to neutrality. Netscape set up a group of 'six founders' when releasing its browser under a GPL-like license, but granted special rights to Netscape and its development partners. Another interesting recent development is the creation of Collab.Net, a for-profit company run by well-respected open source advocates. The company, which has about two dozen clients including HP, Sun and Oracle, is an interface between commercial vendors and the open source community and provides the leadership for open source projects initiated by the release of formerly proprietary code.

While the creation of neutral intermediaries with two-sided reputations is not new, little attention has been devoted to them in economic theory. Furthermore, intermediaries such as Collab.Net seem to differ from existing ones. It exerts leadership and has relatively few clients, in contrast with a rating agency that provides a neutral bridge between bond issuers, investment banks, and investors. Unlike venture capitalists, who reconcile the interests of start-up entrepreneurs with those of institutional investors, the organization enjoys neither strong control rights nor powerful monetary incentives associated with substantial equity stakes.

3.2. Legal aspects

The implications of alternative open source licenses are yet untested. Some argue that more liberal licenses (such as the BSD license, which allows modifications to be commercialized and kept proprietary) permit the 'hijacking' of the open source projects by for-profit companies. Others argue that even some more restrictive licenses (such as the GPL license) still allow the development of proprietary platforms: for example, a successful proprietary office suite for Linux could appropriate the existing value by developing APIs to which programmers would write. For this reason, the most restrictive licenses require that any code that touches an open-source program be made available freely. The problem here is that it is often unclear what the term 'touches' means. Such ambiguity may well discourage commercial companies from porting their proprietary code to open source programs. More theorizing is needed in this respect, as is true much more generally about intellectual property in the software industry.

3.3. Sociological aspects

Last, the discussion of individual incentives raises a host of interesting questions. First, the two delayed incentives, career concerns and ego gratification, are very much driven by the same factors (visibility to a specific audience). It would be interesting to empirically assess the relative importance of these two

factors. Second, programmers' incentives give rise to an interesting form of 'network externalities' or 'strategic complementarities', as programmers assess the future as well as the current viewership of their contributions. The study of the resulting dynamics would have implications not only to the study of open source development and working, but also to other areas such as the evolution of scientific research across fields. Third, we would like to know more about the interaction between open source and commercial software. As in the case of academic research and private consulting, this interaction will have both positive effects (as we argued above) and negative ones (secrecy, partiality, and shifting career concerns and ego gratification incentives).

We hope empirical and theoretical work will be devoted to these and other exciting questions motivated by the open source movement.

References

Arrow, K., 1962. Economic welfare and the allocation of resources for invention.. In: Nelson, R. (Ed.), The Rate and Direction of Incentive Activity: Economic and Social Factors. Princeton University Press, Princeton, NJ.

Lerner, J., Tirole, J., 2000. The simple economics of open source. Working Paper No. 7600, National Bureau of Economic Research, Cambridge, MA.

Mockus, A., Fielding, R., Herbsleb, J., 2000. A case study of open source software program movement: The Apache server, herbsleb@research.bell-labs.com (accessed December 30, 2000).

Raymond, E., 1999. The Cathedral and the Bazaar. O'Reilly, Sebastopol, CA.

Schumpeter, J., 1942. Capitalism, Socialism and Democracy. Harper, New York.