Industry and Innovation

ISSN: 1366-2716 (Print) 1469-8390 (Online) Journal homepage: https://www.tandfonline.com/loi/ciai20

Open Source Software, Competition and Innovation

Jürgen Bitzer & Philipp J. H. Schröder

To cite this article: Jürgen Bitzer & Philipp J. H. Schröder (2007) Open Source Software, Competition and Innovation, Industry and Innovation, 14:5, 461-476, DOI: 10.1080/13662710701711315

To link to this article: https://doi.org/10.1080/13662710701711315

Published online: 04 Dec 2007.

Industry and Innovation,
Vol. 14, No. 5, 461–476, December 2007

Research Paper

Open Source Software, Competition and Innovation

JÜRGEN BITZER*,** & PHILIPP J. H. SCHRÖDER**

**University of Applied Sciences OOW at Emden, Germany, **Aarhus School of Business, University of Aarhus, Denmark*

ABSTRACT The entry and success of open source software (OSS), for example, Linux's entry into the operating systems market, has fundamentally changed industry structures in the software business. In this paper we explore the process of OSS innovation and highlight the impact of increased competition and different cost structures on innovative activity in the industry, which has been neglected in the literature thus far. In a simple model, we formalize the innovation impact of OSS entry by examining a change in market structure from monopoly to duopoly under the assumption that software producers compete in technology rather than price or quantities. The model takes into account development costs and total cost of ownership, whereby the latter captures items such as network externalities. The paper identifies a pro-innovative effect of both intra-OSS and extra-OSS competition.

KEY WORDS: Open source software, innovation, strategic interaction

1. Introduction

The striking difference between open source software (OSS) and proprietary software is the freely accessible source code of the former. This makes it possible to copy, study, improve and customize the source code. Obviously, this has far-reaching effects on the development process of OSS, which is typified by participation of user-programmers, high knowledge spillovers, contributions by volunteers and reuse of source code.

These particularities have led to a dramatic increase in the number of economic studies on the OSS phenomenon in recent years.[1] Economists have shown particular interest in the motivation of OSS contributors (e.g. Johnson, 2001; Hars and Ou, 2002; Lerner and Tirole,

Correspondence Address: Philipp J. H. Schröder, Aarhus School of Business, University of Aarhus, Department of Economics, Prismet, Silkeborgvej 2, DK-8000 Aarhus C, Denmark. Fax: +45 89486125; Tel.: +45 89486392. Email: psc@asb.dk

[1] For a review of the literature see Rossi (2006).

1366-2716 Print/1469-8390 Online/07/050461–16 © 2007 Taylor & Francis
DOI: 10.1080/13662710701711315

2002; Hertel *et al.*, 2003; Bitzer and Schröder, 2005; Lakhani and Wolf, 2005; Bitzer *et al.*, 2007), the dynamics in and management of programmer communities (e.g. Krisnamurthy, 2002; Franke and Shah, 2003; Shah, 2003; von Krogh *et al.*, 2003; Lerner *et al.*, 2006) and the relation between firms and the programmer community (e.g. Dahlander and Magnusson, 2005, 2006; Rossi and Bonaccorsi, 2006). However, the question of what impact this unusual development method has on innovation activity in the software sector has received surprisingly little attention thus far.

Existing research that is in part related to this fundamental question is that of Bitzer (2004) and Casadesus-Masanell and Ghemawat (2006), who analysed whether commercial enterprises can be expected to compete successfully against emerging no-cost OSS competitors or whether they are likely to eventually be displaced. In either case, decreasing profits of proprietary software producers will lower their ability to invest in R&D activity, thus resulting in slower technological progress in the software industry. However, the impact of OSS on the innovation activity in the software sector was not modelled explicitly.

The papers closest to the present one are those of Economides and Katsamakas (2006a) and Bitzer and Schröder (2006). Economides and Katsamakas (2006a) model the interaction between software platforms and software application providers in two different environments: a pure OSS and a pure proprietary ecosystem. Thus, they do not deal with the interaction of different types of firms, and hence the model is unsuitable to address the issue of a change in market structure and its impact on innovation activity, the central question of the present paper. Economides and Katsamakas (2006a) show that the ranking of investment levels in the platform is ambiguous, but that the level of investment in the application is higher with an OSS-based platform than with a proprietary software-based platform. Furthermore, they establish that the level of investment depends on the strength of the reputation effects, the amount of user-programmers and the total cost of ownership (TCO). Bitzer and Schröder (2006) review pro- and anti-innovative features of the OSS development process and provide some initial insights into software competition in a similar framework as in the present paper. However, their model ignores important issues such as network effects and TCO, and they ignore the crucial differences between asymmetric and symmetric software duopolies that are dealt with in the present paper.

While our paper mirrors several of the above assumptions, it takes a different route and addresses a different question. Our focus is on the impact of OSS competition on innovation activity, and accordingly, we set up a formal model of technology competition by extending the framework proposed in Bitzer and Schröder (2006).[2] We apply a general objective function based on the dissemination of a software product to capture the conflicting motives of OSS contributors and profit-oriented software firms. The paper acknowledges—and explicitly models—the demand for software in such a setting, that is, its dissemination, which, is a matter of neither price nor available quantity, but rather of the level of technological content offered by the software to users, its TCO and the technological level of a competitive software product. Given that the TCO is strongly determined by exogenous factors like network effects, user skills, etc., *technology* is the strategic variable of software competition (Shy, 1996).

[2] Within the formal model we do not examine if the OSS business model as such is pro-innovative, see also the discussion in Section 2.

We examine the impact of increased competition on innovation activity in two ways: first, we analyse how a change in market structure—from monopoly to duopoly—in the software industry changes the innovation activity of both the incumbents and the entrants. Our model thus reflects the threat of competition to formerly monopolistic markets from new OSS products. Second, we analyse how different cost structures influence the decision of agents whether or not to innovate. Thus, picking up the often assumed cost advantage of developing OSS in contrast to proprietary software (e.g. Bitzer, 2004; Casadesus-Masanell and Ghemawat, 2006; Economides and Katsamakas, 2006b). However, departing from preceding studies, we do not assume a no-cost development of OSS. Rather, by analysing major differences between the development process of OSS and that of proprietary software, we are able to point out both cost-saving and cost-raising characteristics of the OSS development process.

From the model, we derive the following results: first, the transition from a monopoly to a duopoly (increased competition) increases the technological levels chosen by the enterprises. Second, these findings apply both to pure OSS markets (intra-OSS competition) as well as to mixed markets (e.g. entry of an OSS firm into the market of a for-profit monopolist; extra-OSS competition). Third, assuming that development and innovation costs of OSS firms are lower than those of for-profit firms, pure OSS duopolies will produce more advanced technologies and thus higher rates of innovation. Fourth, assuming that the pay-off for proprietary software producers is higher (and that this dominates the previous effect), a proprietary software duopoly will feature a higher rate of innovation compared to an OSS duopoly. Fifth, and perhaps one of the more counter-intuitive results of the analysis, we are able to show that a higher TCO—independent of the market structure—triggers firms to set higher technology levels, that is, to innovate. This affects both the firm's own rate of innovation as well as its competitor's rate of innovation. The reason for this effect is that firms, within the logic of software competition, must counterbalance a higher TCO with more advanced technological content in order to maintain a satisfactory level of product dissemination.

Finally, a caveat is in place. While the present paper is positioned in the area of OSS and the software industries, its methods and contribution do link back to a wide branch of the industrial organization literature on innovation on market structure (see, for example, the review on theory by Reinganum, 1989, the review of empirical evidence by Cohen and Levin, 1989, or the more recent review by Sutton, 2007). This line of research is triggered by, or related to, the Schumpeterian arguments for monopoly power. The central contribution of Dasgupta and Stiglitz (1980), linking market structure and innovation activity, still sets the standard. In relation to this literature, our present paper—in contrast to the works that follow Dasgupta and Stiglitz (1980)—treats market structure exogenously and focuses on product not process innovation. The paper perhaps closest to the present work is the seminal paper by Bessen and Maskin (2007), dealing with sequential innovation and related to the software industry. Bessen and Maskin (2007) identify pro-innovative effects stemming from the ability to imitate; thus they address an innovation channel distinct from the market structure effect dealt with in the present paper.

The remainder of the paper proceeds as follows. The following section discusses characteristics of the OSS development process affecting costs. Section 3 presents a simple formal model of software competition and derives results on the impact of increased competition on innovation. Section 4 concludes.

2. The Development Process of Open Source

Thus far, research on the influence of OSS on the software industry has taken a traditional competition approach. Bitzer (2004) used a Hotelling set-up to analyse the entry of OSS into formerly monopolistic markets, and Casadesus-Masanell and Ghemawat (2006) analysed competition between OSS and proprietary software in a dynamic mixed duopoly with demand-side learning. Both papers assumed that OSS is offered for free, and based their arguments in part on the observation that development costs appear not to play a role in the OSS context. Thus, both set up a competition model between a firm with development costs and a competitor with zero development costs. Even though such assumptions certainly capture central aspects of the OSS phenomenon, they disguise the fact that OSS development does indeed create costs—private costs that are borne by the contributors. Although volunteer programmers are not able to pass development costs on to the users of the OSS, these costs still arise and may play a role, for example, in deciding whether to program or what solutions/product to program. Development costs may therefore have a strong influence on the innovation activity, not only for proprietary software firms but also for OSS developers. We refrain from discussing the question of whether an OSS contributor has lower private costs in programming OSS—which include opportunity costs—than those incurred by a programmer employed in a firm. Rather we assume that the private costs of an OSS contributor and those of a programmer paid by a firm are largely comparable, and focus on the cost-saving and cost-raising characteristics of the OSS development process.

Abstracting from the details of single OSS projects, the OSS innovation process is typified by a number of characteristics that are valid for the majority of OSS projects. The analysis reveals that several of these lead to lower development costs and TCO for OSS than for proprietary software, while at the same time some characteristics obviously lead to higher OSS development costs and TCO.

2.1. Cost-Saving Characteristics of OSS Development

With OSS, the accessibility of the source code facilitates the emergence of *knowledge spillovers* within the community. Programmers can read, understand and learn from the programming innovations of other programmers. Therefore, the knowledge diffusion within OSS projects takes place unhampered (cf. Raymond, 2000b; Kogut and Metiu, 2001; Osterloh *et al.*, 2002; Haefliger *et al.*, 2006). Proprietary software by its very nature limits this ability to the group of actual project participants, who have access to only parts of the final software product, depending on their rank and position. Obviously, the positive externalities stemming from existing knowledge spillovers reduce the development costs of OSS.

Furthermore, displaying programming steps has a clear disciplining effect on programmers, as it implies an audience. An audience, as in any job, motivates a programmer to provide cleaner, more efficient and more elegant code, compared to compiled software, which disguises cumbersome or faulty programming steps. Thus, the open source code increases the motivation of OSS contributors. Furthermore, the *high motivation* of the contributing programmers is boosted further by the very nature of voluntarism: programmers usually only work on projects that they enjoy (e.g. Luthiger, 2005; Bitzer *et al.*, 2007). Another factor increasing motivation is the value of signalling which results from one's programming work being published with the author's name. Providers of OSS benefit from being able to signal their programming skills, either through improved prospects on the job market (Raymond, 2000b; Lerner and Tirole, 2002; Leppämäki and Mustonen, 2004a, b) and/or through enhanced reputation within the community of programmers (see Raymond, 2000a; Torvalds and Diamond, 2001). Thus, each programmer is interested in maximizing the signal value of his work by providing high-quality software, and this acts as a motivation to produce higher quality programming work. Undoubtedly this aspect of OSS which boosts programmers' motivation also saves costs.

Boundless cooperation is another important advantage of the OSS innovation process. Because commercial exploitation of the newly developed software is not intended, there is no need to keep new ideas secret and therefore barriers against cooperation do not arise. This results in two factors making the OSS innovation process less cost-intensive. First, as already mentioned above, the unlimited access to the source code leads to high knowledge diffusion. Second, as no commercial interests prevent cooperation between programmers, beneficial combinations of complementary programming skills can be exploited (Lakhani and von Hippel, 2003; Haefliger *et al*., 2006).

The cost-saving impact of the *forking thread* effect has not been discussed in the literature thus far. Forking and branching are terms used to describe the splitting up of OSS projects into rival and competing development streams. The right to modify and distribute a splintered version of an existing OSS project puts every programmer in the position to leave the community and set up a new project, further developing a derived version in an alternative direction. This thread leads to a decision process quite different from that in commercial development procedures. The decision on future development is thus based on democratic principles, where the majority of the most important contributors usually decide (e.g. Lerner and Tirole, 2002; Vujovic and Ulhøi, 2006). This has two important effects on the OSS innovation process. First, it ensures that technological aspects are central to the decision on which direction the further development of an OSS project should go. Second, it means that new innovations are implemented immediately.[3]

Another cost-saving characteristic of the development of OSS is the extensive *code reuse* (e.g. Spinellis and Szyperski, 2004; Haefliger *et al*., 2006). Enabled by the terms of OSS licences (Lerner and Tirole, 2005), everything from a few lines of code to entire program structures may be and are reused in other OSS projects. This kind of "recycling" is of course a procedure that cannot be used in the development of proprietary software. Obviously, the reuse of code can save significant amounts of OSS development costs.

Besides the characteristics discussed, which help to save development costs for OSS as compared to proprietary software, some of them also reduce the *total cost of ownership* (TCO) of OSS. The close connection between users and developers is an important way of reducing the TCO. In fact, user and developer are often one and the same person, since the need for a particular solution is a central motive in starting an OSS project or developing a certain extension to an ongoing OSS. This *user-developer* element has been considered to constitute a major advantage of the OSS development process (see Kuan, 2001; Franke and von Hippel, 2003).[4] But even if the programmer and the user are two different people,

[3] An example of a fork that emerged because a sub-group of programmers felt that new contributions and patches had not been released quickly enough is "The Community OpenORB Project" (cf. http://openorb.sourceforge.net/).

[4] In fact, this phenomenon of user-developers is by no means restricted to the realm of OSS. See, for example, Franke and Shah (2003), who analyse cases of user communities that support innovation activities for sports-related products.

the communication between them takes place very directly because the OSS contributor's name is published and he can be addressed directly. Thus, the communication is not hampered by the anonymity and bureaucracy of sales and support departments or other intermediaries. These close connections make it possible to report bugs and request new features much more quickly, activities commercial enterprises have to invest significant resources for.

2.2. Cost-Raising Characteristics of OSS Development

Besides the aforementioned cost-saving characteristics of OSS resulting from the development process and the openness of the source code, there are, of course, also severe cost-raising characteristics.

Due to the freedom within the various OSS communities, there is a high risk of *redundant development*. Thus, instead of searching for prior art, contributors may prefer to offer a "pretended new" solution as it generates a higher reputation in the community. This often results in the "reinvention of the wheel".

Next, we come to the issue of *unhealthy forking* which happens due to the following main reasons: disputes about direction and further development of the OSS project, change of licence of the OSS project and personal disputes between the developers.[5] Once forking has occurred, it is accompanied by several cost-raising effects. First, the appearance of "forks" ultimately reduces the number of programmers working on each of the forks, thus reducing cost-saving effects like cooperation between contributors, spillover effects and boundless cooperation. Second, forking may lead to incompatible standards, reducing the cost-saving effects via code reuse. Third, a splitting of the programmer community might result in a programmer shortage, creating extra costs of attracting programmers (Krishnamurthy and Tripathi, 2006; Lattemann and Stieglitz, 2006; Vujovic and Ulhøi, 2006).

As in the case of cost-saving characteristics, there are cost-raising characteristics which increase the TCO of OSS. In particular, the *undirected innovation* process leads to higher total costs of use of OSS. Development of particular software components is not guaranteed by the respective OSS communities. For example, certain drivers may not exist for Linux, and Linux users have no influence—apart from suggesting the idea for the driver or paying to have it programmed—on the decision of programmers to actually develop it. Furthermore, because of the absence of a liable coordinating institution, there is no guarantee that existing hardware or software will be supported by the operating system in later versions, or that newly purchased hardware or software will be supported. Of course, due to the openness of the source code, it is possible to order single-unit production of the required software component, but this would cause extra costs, and/or the resulting solution would have to be shared among the whole community through the publication of the source code. In contrast, proprietary software enterprises usually pay close attention to the backward and forward compatibility of their software, as well as to the support of older and newly emerging hardware.

[5] Famous examples of software which were forked are: Berkeley Software Distribution (NetBSD), forked into OpenBSD and FreeBSD, and GNU Compiler Collection (GCC) forked into EGCS.

3. A Model of Software Competition

Competition between a profit-oriented software firm and an OSS developer community follows different rules than the "standard" competition model. First, the incentives of the actors differ. Second, the strategic variables are neither price nor quantity, but rather, technology.[6] Third, we include the TCO of a software as an important decision factor for users. Fourth, the behaviour of the market participants is strongly influenced by an exogenous technological factor.

The biggest challenge in capturing software competition is identifying the objective function of the OSS producer. Yet, the majority of research has focused solely on incentives and motives of OSS developers. It is widely acknowledged that, while commercial firms maximize profits, OSS developers are interested in enhancing their reputation and/or signalling value (e.g. Raymond, 2000a, b; Torvalds and Diamond, 2001; Hars and Ou, 2002; Lerner and Tirole, 2002; Hertel *et al.*, 2003; Bitzer and Schröder, 2005; Lakhani and Wolf, 2005; Bitzer *et al.*, 2007). Hence, even though a profit motive can be ruled out for OSS developers, they nevertheless maximize these other pay-offs. Fortunately, although the incentives of the two types of software producers are different, both incentives are clearly correlated with the dissemination of their respective software product. While commercial firms are interested in increasing their profits by benefiting from decreasing average costs with increasing dissemination of their software, OSS developers benefit from the dissemination of their OSS in terms of enhanced reputation and signalling value (Lerner and Tirole, 2002). And most importantly, the presence of network effects in software explains why both types of software producers will value dissemination. Thus, commercial firms and OSS developers can still be assumed to maximize their respective pay-offs, which depend in turn on the dissemination of their software. To capture both types of software producers, we use a general objective function that can represent both proprietary software producers and OSS developers.

Another important aspect in modelling software competition is that the strategic variable in a software market is neither price nor quantity. Software is an intangible good that can be duplicated at virtually no cost, and in addition, at least one agent (the OSS developer) distributes his product at zero price.[7] Since there can be no talk of either quantity *or* price competition, our paper starts out by formulating an (admittedly unconventional) model in which software providers do not compete in price or quantities, but instead in the technological content of their products. A high level of technology ensures widespread dissemination of the software, which is good for the producer, but also raises the costs of development and maintenance (bug fixing etc.). We apply a broad concept of technology including all properties that influence the user's decision to employ a certain software package. Depending on the type of software, the technology therefore includes characteristics like supported hardware, ease of use/installation, interconnectivity capabilities, range of features, state-of-the-art functions, performance, quality, reliability and so on.

[6] In particular, the emergence of no-cost competitors in the software market has altered the type of competition. In such market segments neither price nor quantity are important competition axes but technology became the crucial determinant (see Bresnahan and Greenstein, 1999; Shy, 2001).

[7] In fact, also the price of most proprietary software is often inessential from a consumer's point of view. The majority of software is sold as pre-installed, thus its ability, reliability, compatibility, etc.—in short its technological content—drives the consumer's decision, while the price is a matter between the soft- and hardware producer.

Thus, the technology of software includes the entire bundle of its technological characteristics (Bessen, 2006), see Bitzer and Schröder (2006) for an earlier version of a duopoly model with the above properties.

Finally, a model of software competition must take into consideration TCO, which includes not only the cost of purchase but also all aspects of the use and maintenance of the software, for example, costs of training users and IT support staff, costs associated with failure or outage of the software (planned and unplanned downtime), administration costs, development costs, costs of overcoming network externalities[8] and switching costs (CSC, 2004). In particular, the latter two are important characteristics of the software market (Schmidt and Schnitzer, 2003); see Farrell and Klemperer (2007) for a comprehensive survey of switching costs and network effects.

These considerations lead us directly to the central difference between our model and a "standard" competition setting. The "value" of a piece of software to a user depends strongly on how up to date its general functionality is or, to put it differently, its "real" technological level. The real technological level depends on how far each technological characteristic of the software is behind the state of the art: the "technological frontier" of that particular aspect of the software.[9] Thus, the technological level of any piece of software is defined in relation to the global technological level, which consists of all the most advanced developments in each aspect of that software at that specific point in time.[10] On the other hand, the global technological level itself is constantly changing. It is set by developments in the globally available technology that influence the demand for or development of software. The global technological level is driven by developments in hardware technology, new applications, new features, consumer demands and so on. The "real" technological level of any piece of software quickly deteriorates as externally determined technological possibilities (processor capacity, application demands, etc.) and consumer demands grow. Therefore, the technology embedded in a developed piece of software must be adjusted in accordance with external (global) technological progress, that is, innovation.

3.1. A Simple Framework

Consider a heterogeneous goods software duopoly. The two firms *a* and *b* service the imperfectly separated market segments *A* and *B*, respectively.[11] Instead of competing on price or quantity, the two firms compete in the technology of their respective software products. Hence the strategic variable is the technological level of the product, or rather, innovation and development. At time *t* the technological level of firm *i*'s software is denoted by τ_t^i, $i=a, b$. Further, the global technology level, T_t, represents hardware advances, new applications and changes in consumer demand.

[8] It is always possible to overcome network externalities by corresponding investments, for example, paying someone to program a missing piece of application software. Of course, the required investment costs might be exorbitantly high. However, network externalities can be interpreted as a component of the TCO.

[9] Shy (1996) similarly argues that not every newly developed technology is adopted leading to the same conclusion.

[10] As no single software product is at the technological frontier in terms of every one of its technological dimensions, the technological level of any specific piece of software must always be lower than the global technological level.

[11] Even though we use the term "firm" here, it is applied in the sense of agency, in that it may either refer to traditional proprietary software firms or OSS developer communities. Thus we are not referring to for-profit firms that base their business on OSS.

The demand for software—or rather, the dissemination of the two software products—is assumed to be symmetric and to depend on the technological advance/ability of the software, on the interdependence between the two market segments, that is, the two products are imperfect substitutes and the presence of network externalities which influence the TCO. Dissemination, s_t^i, of the two products *a* and *b* at time *t* is represented by $s_t^i = \max\{0, q_t^i\}$, $i = a, b$, where

$$q_t^a = \left(\frac{\alpha}{2} + \beta\right) \frac{\tau_t^a}{T_t} - \frac{\beta \tau_t^b}{T_t} - k_a \qquad (1)$$

$$q_t^b = \left(\frac{\alpha}{2} + \beta\right) \frac{\tau_t^b}{T_t} - \frac{\beta \tau_t^a}{T_t} - k_b. \qquad (2)$$

Parameter $\beta > 0$ is a measure of the substitutability of the two products and α is a positive constant. The parameters k_a, $k_b > 0$ represent TCO, implying that dissemination falls in this cost. The expressions τ_t^i / T_t represent the "real technology level" of firm *i*: namely, how advanced the technological capability of the software product is in relation to the hardware ability, consumer demands etc. Thus (1) and (2) state that the extent to which a certain software producer's product is distributed/sold depends positively on its own real technology level and negatively on the competitor's real technology level. The presence of β and k_i imply that the system captures network externalities in two ways: first directly, as TCO may be higher for software with a small user base,[12] and secondly indirectly, via the linkage of substitutability between the two products.[13] To illustrate this property of the system, consider that after substituting out τ_b in (1) via (2) one finds $\partial q_t^a / \partial q_t^b < 0$, thus an increase in the dissemination in one software has a negative effect on the dissemination of the other software. In order to focus on issues of strategic interaction among agencies, the present approach only captures network effects on the demand side (see also Farrell and Klemperer, 2007). In contrast, the production side externalities present in OSS, such as the network driven bandwagon dynamics of programmers joining projects (see Bitzer and Schröder, 2005), are ignored. Equations (1) and (2) also take account of the effects of external technological development in T_t which devalues the real technology level and dissemination of both software products.[14] Finally, non-negative dissemination levels in equilibrium require $k_i < \alpha \tau_t^i / 2T_t$, $i = a, b$; if the TCO violates this condition, dissemination of the software is zero (by $s_t^i = \max\{0, q_t^i\}$).

Although the model avoids the notion of price or quantity competition, the principles of maximization are still applicable. Assume that firms derive some pay-off from each unit of software distributed/sold. In particular there is a gain ρ, which could represent the reputational or signalling value for an OSS producer or more conventionally, the per-unit

[12] For example, adaptation of the software in terms of programming interfaces or adaptation to users, or training of users is usually more expensive if the software has a small user base.

[13] Network effects determine the substitutability as it might be difficult to change from a software with a large user base to one with a small one. Because one might loose positive network effects, for example, changing from MS Office to KOffice might prevent or hamper the exchange of documents with outsiders.

[14] Or put differently, if both firms opt for technological stand-still, global developments will eventually reduce the dissemination of the two products to zero.

monetary reward for a proprietary software firm. There is also a cost, C_i, which is assumed to depend proportionally on the technology level of the product. Thus, a high real technology level is associated with high development and maintenance costs, for instance, larger support or hotline costs, costs of bug fixing, or indeed distribution and production costs. The latter in particular can be seen as driven by the fact that programmers working on a more advanced software product are more expensive than those working on an inferior software project. Postulating $C_i = c_i(\tau_t^i / T_t)$, the gain function for firm i can be stated as:

$$g_t^i = q_t^i \left(\rho - c_i \frac{\tau_t^i}{T_t} \right), \quad i = a,b \tag{3}$$

where the participation constraint requires non-negative gain, namely, $\rho > c_i(\tau_t^i / T_t)$.

How does the situation of software duopoly compare to that of a software monopolist? Assume that a monopolist is servicing both market segments *A* and *B* with the respective software products *a* and *b*. The gain functions for the monopolist are identical to those formulated in (3). Yet the monopolist is aware of the interaction of the two markets, represented by β in (1) and (2), and takes this fact into account. In particular, due to symmetry, a monopolist, *M*, realizes that $\tau_t^{Ma} = \tau_t^{Mb}$. Rewriting the monopolist's gain function for market *i* after setting in (1) and (2), respectively, gives

$$g_t^{Mi} = \left(\frac{\alpha}{2} \frac{\tau_t^{Mi}}{T_t} - k_i \right) \left(\rho - c_i \frac{\tau_t^{Mi}}{T_t} \right), \quad i = a,b. \tag{4}$$

Since both markets behave identically, in subsequent analysis it suffices to consider only one of the market segments.

3.2. *Innovation—Setting Technology Levels*

We start out by considering the symmetric case where firms are identical in the sense that $k_a = k_b = k$ and $c_a = c_b = c$. This case is suitable for an analysis where both market segments are serviced by identical types of firms—either two OSS competitors (forking) or two proprietary software producers. In the case of a software duopoly, where firms behave non-cooperatively and simultaneously have to choose their respective technology levels to maximize (3), the first-order condition for firm *a* after substituting in q_t^a from (1) becomes

$$\tau_t^a = \frac{\rho T_t}{2c} + \frac{\beta \tau_t^b + k T_t}{\alpha + 2\beta} \tag{5}$$

and similarly for firm *b*.

Given that both firms expect all future technology levels T_{t+j} ($j=1, 2, \ldots$) to be equal to T_t, then the Nash equilibrium technology levels at time *t* are

$$\tau_t^a = \tau_t^b = \frac{T_t(\rho(\alpha + 2\beta) + 2ck)}{2c(\alpha + \beta)}. \tag{6}$$

Lemma 1. *The technology level set by a symmetric software duopoly increases for a higher pay-off, ρ, a falling cost, c, a higher TCO, k, and a higher degree of substitutability, β.*
Somewhat surprisingly, the derivative of (6) with respect to *k* is positive. The intuition for this result is that software—if the TCO is higher—needs to offer a more advanced

technological content in order to still attract users and to compensate them for the higher *k*. Thus, if we assume that an OSS duopoly has a higher TCO *k* compared to a for-profit software duopoly, then the above result states that the OSS duopoly will result in higher technology levels but also less total dissemination of software (following directly from (1) and (2)). Thus, it is a more specialized, expert-type application. Similarly, if we assume that an OSS duopoly has lower costs *c* compared to a for-profit software duopoly, then Lemma 1 states that an OSS duopoly will produce a higher technological level. In contrast, a duopoly with a higher pay-off ρ, which may be the case for for-profit firms, will settle for a higher technology level. Finally, once the two software products become more homogeneous (higher β) the strategic interaction in a market with network externalities triggers the software duopoly firms to set higher technology levels.

While the above results all relate to the symmetric case, in the sense that firms with identical cost structures interact, it is important to derive the asymmetric case—capturing important aspects of the encounter between a for-profit firm and an OSS producer. In a software duopoly with heterogeneous development costs, c_a and c_b, and heterogeneous total costs of use, k_a and k_b, firm *a*'s Nash equilibrium technology level becomes

$$\tilde{\tau}_t^a = \frac{\rho T_t(\alpha+2\beta)(c_a\beta + c_b(\alpha+2\beta))}{2c_a c_b(\alpha+\beta)(\alpha+3\beta)} + \frac{T_t(k_b\beta + k_a(\alpha+2\beta))}{(\alpha+\beta)(\alpha+3\beta)} \tag{7}$$

and similar for firm *b*. It is easy to verify that $\partial\tilde{\tau}_t^a/\partial c_a, \partial\tilde{\tau}_t^a/\partial c_b < 0$ and $\partial\tilde{\tau}_t^a/\partial k_a, \partial\tilde{\tau}_t^a/\partial k_b > 0$.[15] Thus, we obtain two results:

Lemma 2. *A reduction in the development cost c_i of one firm in an asymmetric software duopoly increases the individual technology levels set by both firms.*

Lemma 3. *A reduction in the TCO k_i of one firm in an asymmetric software duopoly decreases the individual technology levels set by both firms.*

Put differently, a low-cost competitor *b* increases the technology level set by firm *a*, and an increase in both the firm's own and the competitor's TCO triggers a higher technology level.[16] In terms of Microsoft versus OSS, Lemma 2 implies that the entry of a low-cost competitor pushes up the technology level of the for-profit firm beyond the level that would have resulted from a for-profit entry with higher costs, c_b. With respect to the impact of the TCO parameters k_a and k_b, Lemma 3 implies that—assuming that OSS typically has fairly high TCO, not least due to network externalities, switching costs, etc.—entry of OSS competition may be associated with higher technology levels. However, it is important to note that the increase in technology levels in response to higher total costs of use is clearly associated with an overall reduction in the total dissemination of software—put differently, fewer users will choose a software with a higher *k*.

Compare the above findings to a software monopolist maximizing (4), and expecting all future technology levels T_{t+j}, j=(1, 2, ...) to be equal to T_t. Obviously, since production is within one firm that services two market segments, we have to return to the assumptions c_a=c_b=c and k_a=k_b=k. Such a monopolist has first-order conditions for market segments *A*

[15] Obviously all other properties laid out in Lemma 1 directly extend to the asymmetric case.

[16] The intuition for the latter effect is, of course, that a higher k_b forces firm b *to offer a higher technology level such as to compensate users for the TCO, and this in turn triggers firm* a *via the strategic interaction to respond with a higher technology level itself.*

and B that define the optimal technology level as

$$\tau_t^{Ma} = \tau_t^{Mb} = \frac{\rho T_t}{2c} + \frac{Tk}{\alpha}. \tag{8}$$

Lemma 4. *The technology level set by a software monopolist increases for higher pay-off, ρ, a falling cost, c, a higher TCO, k, but is independent of the degree of substitutability, β.*

Thus, given that an OSS developer can be fairly assumed to have a lower cost c, and a higher TCO, k, an OSS monopolist will provide a higher technology level than a for-profit monopolist. Yet, if the for-profit monopolist has a higher pay-off ρ compared to the pay-off for the OSS developer, then the reverse conclusion applies.

Comparing (6) to (8) leads to the following central proposition.

Proposition 1. *Given a global technology level T_t, each firm i=a, b in a software duopoly sets a technology level τ_t^i for its respective software product that exceeds the technology level set by a software monopolist, τ_t^{Mi}.*

Proof. $\tau_t^i - \tau_t^{Mi} = T_t \beta((\alpha\rho - 2ck)/(2c\alpha(\alpha+\beta))) > 0$ follows from non-negative dissemination levels in equilibrium ($k < \alpha\tau_t^i / 2T_t$, $i=a$, b) and the participation constraint ($\rho > c(\tau_t^i / T_t)$).

Proposition 1 states that a monopolist will choose a lower technology level than a software duopoly.[17] The monopolist, in contrast to the duopoly, takes into account the externality that a high technology level in one software segment has on the dissemination of other software products in his portfolio. Proposition 1 carries an important message concerning claims that competition can be harmful in technology and research-intensive industries. If firms do indeed compete in technology, then competition—moving from a monopoly situation to a duopoly situation—triggers innovation activity and arrives at a higher technological level; thus competition has a pro-innovative effect. This beneficial impact of entry occurs no matter whether the entry takes place in a commercial software market or in an existing OSS market, for instance, the case of forking of an OSS project.

Proposition 1 contains the intuitively compelling insight that, in a duopoly setting, not only do firms innovate their product in order to optimize their technology position with respect to the outside technological development, they also innovate in anticipation of the other firm's innovation activity. The size of this pro-innovative effect from competition depends on the various parameters of the model.

Corollary 1. *The pro-innovative effect of competition laid out in Proposition 1 increases for higher pay-off, ρ, a falling cost, c, a falling TCO, k, and an increasing degree of substitutability, β.*

Proof. Follows from the derivatives of $\tau_t^i - \tau_t^{Mi}$ with respect to ρ, c, k and β. In particular the impact of β is closely related to network externalities, for instance, the pro-innovative effect is stronger the closer substitutes the goods are, that is, the closer competitors the two firms are.

The model suggests a number of testable hypotheses and themes suitable for future research. From the above results, one would expect to observe a higher technological level

[17] Notice that Proposition 1 is purposefully derived under the assumptions $c_a = c_b = c$ and $k_a = k_b = k$, such that our results stem solely from the strategic interaction of firms, that is, competition, rather than from the possible cost advantages that an OSS competitor may or may not have.

for OSS markets and for software markets that experience entry or forking. The model further suggests that in mixed markets where one competitor is a proprietary software and the other is an OSS developer, the chosen technological level is higher than in a pure proprietary software duopoly, but lower than in a pure OSS duopoly. Thus, within the present framework, evaluated in terms of the technology level, the pure OSS duopoly would dominate all other market structures treated in the paper. However, most importantly, the model also makes clear predictions on software dissemination. In particular, based on common assumptions about development costs and TCO, the model suggests that OSS software is expected to be used more frequently as fairly narrow-expert applications, where there is a high technological content level but a small user base.

4. Conclusion

The paper analyses the influence of entry and competition by open source software (OSS) on innovation and technological progress in software markets. The best-known example of such an event is the entry of Linux into the market for server operating systems. Some observers fear that the technological progress in software technology will slow or even stop altogether as a consequence of entry of a low-cost OSS competitor into former highly concentrated (monopolistic) markets.

Departing from former research on OSS, we explicitly model the influence of competition on the decision to innovate. We examine the impact of increased competition on innovation activity in two ways: first, we analyse how a change in market structure—from monopoly to duopoly—in the software industry changes the innovation activity of both the incumbents and the entrants. Secondly, we analyse how different cost structures influence the decision of the agents to innovate or not. Thus, we re-examine the often assumed cost advantage of OSS over proprietary software. However, departing from preceding studies, we do not assume a no-cost development of OSS. Reviewing the differences in the development process of OSS and proprietary software, we find both cost-saving and cost-raising characteristics of the OSS development process.

We set up a simple model of software competition where producers compete in technology rather than price or quantity. Within the model, the development decision of the firms regarding how to set the technology level of their software (innovate) is examined. We find that the move from monopoly to duopoly always increases the technology level and thus the level of innovation chosen by the enterprises. Thus, OSS entry has a positive impact on firms' willingness to innovate and heightens the overall technological level in the industry. Furthermore, under the assumption that the development and innovation costs of OSS firms are lower than those of commercial firms, the model implies that in terms of technology levels and rate of innovation, a pure OSS duopoly dominates monopolies (either proprietary or OSS), pure commercial duopolies and mixed duopolies (e.g. one OSS firm and one for-profit firm). Put differently, competition is still good, also when the product—in this case software—is knowledge-intensive.

Our model also raises several new questions that have to be left to future research. First, the question of whether the structure and organization of the OSS development process leads to higher and/or better output has to be answered empirically. Second, our model underlines the importance of the TCO for the purchase decision, but, the jury is still out on whether or not the TCO of OSS is higher or lower than that of proprietary software. Finally, almost completely neglected here is the question of if and to what extent knowledge spillovers occur during the OSS development process.

Acknowledgements

The authors thank Roger Sherman and Zhentang Zhang for valuable comments. Previous versions of this paper have been presented at the International Industrial Organization Conference (IIOC) at Northwestern University, Chicago, the Annual Meeting of the Canadian Economic Association (CEA), Toronto and the annual conference of the European Association for Research in Industrial Economics (EARIE) at Humboldt University, Berlin. The usual disclaimer applies.

References

Bessen, J. (2006) Open source software: free provision of complex public goods, in: J. Bitzer & P. J. H. Schröder (Eds) *The Economics of Open Source Software Development*, pp. 57–82 (Amsterdam: Elsevier).

Bessen, J. and Maskin, E. (2007) Sequential innovation, patents and imitation, *RAND*, forthcoming.

Bitzer, J. (2004) Commercial versus open source software: the role of product heterogeneity in competition, *Economic Systems*, 28(4), pp. 369–381.

Bitzer, J. and Schröder, P. J. H. (2005) Bug-fixing and code-writing: the private provision of open source software, *Information Economics and Policy*, 17(3), pp. 389–406.

Bitzer, J. and Schröder, P. J. H. (2006) The impact of entry and competition by open source software on innovation activity, in: J. Bitzer & P. J. H. Schröder (Eds) *The Economics of Open Source Software Development*, pp. 219–246 (Amsterdam: Elsevier).

Bitzer, J., Schrettl, W. and Schröder, P. J. H. (2007) Intrinsic motivation in open source software development, *Journal of Comparative Economics*, 35(1), pp. 160–169.

Bresnahan, T. F. and Greenstein, S. (1999) Technological competition and the structure of the computer industry, *Journal of Industrial Economics*, 47(1), pp. 1–40.

Casadesus-Masanell, R. and Ghemawat, P. (2006) Dynamic mixed duopoly: a model motivated by Linux vs. Windows, *Management Science*, 52(7), pp. 1072–1084.

Cohen, W. M. and Levin, R. C. (1989) Empirical studies of innovation and market structure, in: R. Schmalensee & R. D. Willig (Eds) *Handbook of Industrial Organization* (Amsterdam: Elsevier).

CSC (Computer Sciences Corporation) (2004) Open source: open for business. Available at: http://www.csc.com/features/2004/uploads/LEF_OPENSOURCE.pdf (accessed 25 July 2006).

Dahlander, L. and Magnusson, M. G. (2005) Relationships between open source software companies and communities: observations from Nordic firms, *Research Policy*, 34(4), pp. 481–493.

Dahlander, L. and Magnusson, M. G. (2006) Business models and community relationships of open source software firms, in: J. Bitzer & P. J. H. Schröder (Eds) *The Economics of Open Source Software Development*, pp. 111–130 (Amsterdam: Elsevier).

Dasgupta, P. and Stiglitz, J. (1980) Industrial structure and the nature of innovation activity, *The Economic Journal*, 90, pp. 266–293.

Economides, N. and Katsamakas, E. (2006a) Linux vs. Windows: a comparison of application and platform innovation incentives for open source and proprietary software platforms, in: J. Bitzer & P. J. H. Schröder (Eds) *The Economics of Open Source Software Development*, pp. 207–218 (Amsterdam: Elsevier).

Economides, N. and Katsamakas, E. (2006b) Two-sided competition of proprietary vs. open source technology platforms and the implications for the software industry, *Management Science*, 52(7), pp. 1057–1071.

Farrell, J. and Klemperer, P. (2007) Coordination and lock-in: competition with switching costs and network effects, in: M. Armstrong & R. Porter (Eds) *Handbook of Industrial Organization* (Amsterdam: Elsevier).

Franke, N. and Shah, S. (2003) How communities support innovative activities: an exploration of assistance and sharing among end-users, *Research Policy*, 32(1), pp. 157–178.

Franke, N. and von Hippel, E. (2003) Satisfying heterogeneous user needs via innovation toolkits: the case of Apache security software, *Research Policy*, 32(7), pp. 1199–1215.

Haefliger, S., von Krogh, G. and Spaeth, S. (2006) Knowledge reuse in open source software, Working Paper, ETH Zurich.

Hars, A. and Ou, S. (2002) Working for free? Motivations for participating in open-source projects, *International Journal of Electronic Commerce*, 6(3), pp. 25–39.

Hertel, G., Nieder, S. and Herrmann, S. (2003) Motivation of software developers in open source projects: an Internet-based survey of contributors to the Linux kernel, *Research Policy*, 32(7), pp. 1159–1177.

Johnson, J. P. (2001) Open source software: private provision of a public good, *Journal of Economics and Management Strategy*, 11(4), pp. 637–662.

Kogut, B. and Metiu, A. (2001) Open-source software development and distributed innovation, *Oxford Review of Economic Policy*, 17(2), pp. 248–264.

Krishnamurthy, S. (2002) Cave or community? An empirical examination of 100 mature open source projects, *First Monday*, 7(6), Available at www.firstmonday.org.

Krishnamurthy, S. and Tripathi, A. K. (2006) Bounty programs in free/libre/open source software (FLOSS), in: J. Bitzer & P. J. H. Schröder (Eds) *The Economics of Open Source Software Development*, pp. 165–184 (Amsterdam: Elsevier).

Kuan, J. (2001) Open source software as consumer integration into production, Working Paper. Available at http://ssrn.com/abstract=259648 (dated 29 July 2004).

Lakhani, K. R. and von Hippel, E. (2003) How open source software works: "free" user-to-user assistance, *Research Policy*, 328(7), pp. 923–943.

Lakhani, K. R. and Wolf, R. G. (2005) Why hackers do what they do: understanding motivation effort in free/open source software projects, in: J. Feller, B. Fitzgerald, S. Hissam & K. R. Lakhani (Eds) *Perspectives on Free and Open Source Software*, pp. 3–21 (Cambridge, MA: MIT Press).

Lattemann, C. and Stieglitz, S. (2006) Coworker governance in open-source projects, in: J. Bitzer & P. J. H. Schröder (Eds) *The Economics of Open Source Software Development*, pp. 149–164 (Amsterdam: Elsevier).

Leppämäki, M. and Mustonen, M. (2004a) Signalling and screening with open source programming, Helsinki School of Economics Working Papers, No. W-377.

Leppämäki, M. and Mustonen, M. (2004b) Signalling with externality, HECER—Helsinki Center of Economic Research Discussion Papers, No. 22.

Lerner, J. and Tirole, J. (2002) Some simple economics of open source, *Journal of Industrial Economics*, 50(2), pp. 197–234.

Lerner, J. and Tirole, J. (2005) The scope of open source licensing, *Journal of Law, Economics and Organization*, 21(1), pp. 20–56.

Lerner, J., Pathak, P. A. and Tirole, J. (2006) The dynamics of open-source contributors, *American Economic Review*, AEA Papers and Proceedings 96(2), pp. 114–118.

Luthiger, B. (2005) Fun and development, in: M. Scotto & G. Succi (Eds) *Proceedings of the First International Conference on Open Source Systems*, pp. 273–278.

Osterloh, M., Kuster, B. and Rota, S. (2002) Open source software production—climbing on the shoulders of giants, Working Paper, University of Zürich.

Raymond, E. S. (2000a) Homesteading the noosphere, Revision 1.22, 24 August, first version 1998.

Raymond, E. S. (2000b) The cathedral and the bazaar, Revision 1.51, 24 August, first version 1997.

Reinganum, J. (1989) The timing of innovation: research, development, and diffusion, in: R. Schmalensee & R. D. Willig (Eds) *Handbook of Industrial Organization* (Amsterdam: Elsevier).

Rossi, M. A. (2006) Decoding the free/open source software puzzle: a survey of theoretical and empirical contributions, in: J. Bitzer & P. J. H. Schröder (Eds) *The Economics of Open Source Software Development*, pp. 15–56 (Amsterdam: Elsevier).

Rossi, C. and Bonaccorsi, A. (2006) Intrinsic motivations and profit-oriented firms in Open Source software. Do firms practise what they preach?, in: J. Bitzer & P. J. H. Schröder (Eds) *The Economics of Open Source Software Development*, pp. 83–110 (Amsterdam: Elsevier).

Schmidt, K. M. and Schnitzer, M. (2003) Public subsidies for open source? Some economic policy issues of the software market, *Harvard Journal of Law & Technology*, 16(2), pp. 474–505.

Shah, S. K. (2003) Understanding the nature of participation & coordination in open source software development communities, Dissertation, Chapter 4, MIT Sloan School of Management.

Shy, O. (1996) Technology revolutions in the presence of network externalities, *International Journal of Industrial Organization*, 14, pp. 785–800.

Shy, O. (2001) *The Economics of Network Industries* (Cambridge: Cambridge University Press).

Spinellis, D. and Szyperski, C. (2004) How is open source affecting software development?, *IEEE Software*, pp. 28–34.

Sutton, J. (2007) Market structure: theory and evidence, in: M. Armstrong & R. Porter (Eds) *Handbook of Industrial Organization* (Amsterdam: Elsevier).

Torvalds, L. and Diamond, D. (2001) *Just for Fun: The Story of an Accidental Revolutionary* (New York: HarperBusiness).

von Krogh, G., Spaeth, S. and Lakhani, K. R. (2003) Community, joining, and specialization in open source software and innovation: a case study, *Research Policy*, 32(7), pp. 1217–1241.

Vujovic, S. and Ulhøi, J. P. (2006) An organizational perspective on free and open source software development, in: J. Bitzer & P. J. H. Schröder (Eds) *The Economics of Open Source Software Development*, pp. 185–206 (Amsterdam: Elsevier).