WORKING PAPER 2005-12

WP

Jürgen Bitzer and Philipp J.H. Schröder

The Impact of Entry and Competition by Open Source Software on Innovation Activity

International Business Section

The Impact of Entry and Competition by Open Source Software on Innovation Activity

Jürgen Bitzer* Philipp J.H. Schröder†

December 2005

Abstract

This paper presents the stylized facts of open source software innovation and provides empirical evidence on the impact of increased competition by OSS on the innovative activity in the software industry. Furthermore, we introduce a simple formal model that captures 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 paper identifies a pro-innovative effect of OSS competition.

Keywords: open source software, innovation, strategic interaction.

*Free University Berlin, Germany.

†Aarhus School of Business, Denmark.

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.

1 Introduction

Open Source Software (OSS) has increased competition in the software industry, as famously and involuntarily acknowledged by a major incumbent software firm:

> "OSS poses a direct, short-term revenue and platform threat to Microsoft, ... OSS has benefits that are not replicable with our current licensing model and therefore present a long term developer mindshare threat .... Recent case studies provide very dramatic evidence that commercial quality can be achieved/exceeded by OSS projects."
>
> (Vinod Valloppillil, Microsoft Software Engineer quoted in: Halloween Documents, 1998).[1]

Major OSS projects have, captured significant market shares from their proprietary competitors – commercial software producers. Linux, for example, accounts for a 38% share of the server operating system market (Bitzer 2004) and is supported by major hardware firms such as SUN, Compaq/HP, IBM and Siemens. The OSS web server 'Apache' captured a market share of 68% by December 2004.[2] Recently, the open source web browser 'Firefox' has been able to snatch a 5% market share from Microsoft's Internet Explorer over the six-month period from May 2004 to October 2004 (Festa 2004). Most importantly, none of these products have ever been sold in a retail shop.[3] Furthermore there are literally millions of people involved in programming and improving OSS and the huge majority of them have never received any form of monetary reward for their efforts.

In this paper we examine the entry of this unusual production method into the software industry and its impact on innovation activity in the sector. In particular, the emergence of OSS as competing products in formerly highly

[1] The origins of the Halloween documents are somewhat foggy. Allegedly, the documents are an internal Microsoft memo that was leaked to the prominent OSS advocate Eric Raymond. After an initial dispute about these documents, Microsoft eventually acknowledged their authenticity but dismissed them as low-level engineering studies. More background information on these documents can be found on the Open Source Initiative home page [http://opensource.mirrors.ilisys.com.au/halloween/].

[2] Cf. information on [http://news.netcraft.com/archives/web_server_survey.html].

[3] Yet, services surrounding OSS products may be sold commercially. To be more precise – and as will be elaborated in Section 2 below – OSS distributors may charge such fees for their services as copy fees, maintenance, and individualizations of software, but not for the OSS it self. Accordingly, any individual equipped with sufficient computer skills can download, install, and operate any OSS product free of charge.

concentrated software markets (extra-OSS competition) and the constant threat of forking and branching of OSS projects (intra-OSS competition) has raised fears of a potential anti-innovative effect.[4] These concerns about a potentially anti-innovative effect of OSS focuses on two main issues. First, the emergence of a no-cost competitor on the software market raised the question of whether commercial enterprizes will be able to compete successfully or if they will eventually be displaced (Casadesus-Masanell and Ghemawat 2003, Bitzer 2004). In either case, decreasing profits of commercial software producers will lower their ability to invest in R&D activity, thus resulting in slower technological progress in the software industry. Second, an anti-innovative effect of OSS may result if its development process is less efficient than that of commercially organised software:

> "The OSS development model leads to a strong possibility of unhealthy 'forking' of a code base, resulting in the development of multiple incompatible versions of programs, weakened interoperability, product instability ..."
>
> (Craig Mundie, Microsoft Senior Vice President, talk given at New York University Stern School of Business, cited after Microsoft 2001).

Both arguments imply that the emergence and success of OSS leads to diminished technological progress in the software industry overall. While the first argument implies a reduction in commercial enterprises' innovation activity, the second suggests a waste of resources in OSS production per se, for example due to reductions in scale. Both arguments are founded on the belief that increased competition in knowledge-intensive industries – like the software business – might harm innovation activity. The present paper examines this issue in detail. First, we present background information on the OSS development process and the pro- and anti-innovative forces inherent in this development model. Second, we compile and review empirical evidence on the success and speed of innovation in software market segments where the entry of OSS has created new competitive situations, with both commercial and other OSS producers. From this review, we conclude that there is no evidence of an anti-innovative impact. On the contrary, it appears that the entry of OSS into commercial segments of the market and the forking that is occurring in some ongoing OSS projects are associated with increased

[4]The terms forking and branching refer to the risk that groups of OSS programmers may splinter off an existing project and develop the project in their own direction, thus ultimately competing for the same user groups and customer base. By permitting this possibility, OSS contains the potential for competition in its basic structures.

innovation. Third, in a formal model we examine the impact of increased competition on innovation activity; i.e. a change in market structure from monopoly to duopoly in the software industry. The model also shows a pro-innovative effect of OSS competition.

In the formal model we propose a general objective function based on the dissemination of a software product and its technological level, where the firm-specific technological level is driven by innovation activity.[5] The paper acknowledges – and explicitly models – that in such setting the demand for software, i.e. its dissemination, is a matter of neither price nor available quantity, but rather of the technological content (level) that the software offers to users. Thus, *technology* is the strategic variable of software competition. 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, then pure OSS duopolies will produce more advanced technologies and thus higher rates of innovation.

There exists a growing literature on OSS, see Rossi (2006) for a comprehensive literature survey. The history of OSS is documented, in for example, Stallmann (1999), Rosenberg (2000), Torvalds and Diamond (2001), Feller and Fitzgerald (2002) and Hars and Ou (2002). The existing literature highlights a variety of motives for OSS programmers to participate and innovate – ranging from the need for a particular software solution (user-developers) to intrinsic motives such as fun and play – and emphasizes the importance of community structures and product structures (e.g. modularity) for the success of OSS: see for example Kuan (2001), Franke and von Hippel (2003), von Krogh, Spaeth and Lakhani (2003), Shah (2003), Baldwin and Clark (2003), Narduzzo and Rossi (2004), Ulhøi (2004) and O'Mahony and Ferraro (2004). Other approaches to the OSS issue are, for example, Lerner and Tirol (2002), who emphasize the role of signaling programming skills; Mustonen (2003), who presents a model of OSS licensing schemes and their effects on programmer labour markets; Bitzer (2004) and Casadesus-Masanell and Ghemawat (2003), who deal with the competitive impact of challenging a commercial software firm with an OSS alternative in theoretical terms; and

[5]Our paper differs from the industrial organization literature in its assessment of the influence of competition on innovation activity – as initially raised by Schumpeter (1934, 1942) and Arrow (1962). Our paper treats strategic decisions on non-drastic product innovation as driven, inter alia, by exogenous technological advancements.

Johnson (2001) and Bitzer and Schröder (2005), Bitzer et al. (2004), who model OSS development as a private provision of a public good situation.

Yet, while there is this substantial and growing economics and business literature on the organizational, motivational, and product-structural features of OSS development, the impact of intra- and extra-OSS competition on innovation has not yet been examined. There are, however, several papers discussing the innovation process in OSS itself, which thus relate to the present work. Narduzzo and Rossi (2004) examine the role that modular product architecture has as a device for facilitating rapid and constant innovation in OSS and highlight how OSS projects utilize these features. Ulhøi (2004) compares open source innovation (going beyond the realm of software) versus the proprietary knowledge model of innovation, and emphasizes the challenge that OSS innovation poses to traditional economic theories of innovation. Finally, the role of user-developers in innovative activity has been emphasized by several authors, see e.g. Kuan (2001) or Franke and von Hippel (2003), and has also been identified outside the production of software (Franke and Shah 2003).

The remainder of the paper proceeds as follows. The following section provides an introduction to OSS pro- and anti-innovative features. In Section 3, we collect and discuss available empirical evidence on innovative activity in the software industry since the arrival of OSS. Section 4 presents a simple formal model of software competition and derives results on the impact on innovation from increased competition. Section 5 concludes.

2 Open Source Innovation

Three frequently posed questions concerning OSS are. How does OSS development work? Who is willing to programm OSS and why? And how are the internal structures of OSS projects organized? In the following, we briefly summarize the answers put forward in the literature so far.[6] OSS software is programmed by volunteers who do voluntarily engage in a project and as a rule do not receive any pay for their efforts. While these programmers are only loosely associated in community structures, 'tacit' knowledge of the project is passed on as explicit knowledge through project websites, and more importantly, it is embedded in the source code of the software itself. Individual OSS projects can attract substantial numbers of programmers. One of the biggest OSS projects is Linux, with its estimated 10,000 active developers. OSS employs a modular product architecture. Agents develop

[6]See Rossi (2006), for a survey of the literature, and our own Bitzer and Schröder (2006), for a short introduction to OSS development.

or improve the available OSS software by creating complementary software modules, thus repeatedly contributing to the product (on the issue of modular versus integral product architecture, see Ulrich (1995)). The importance of a modular product architecture in OSS has been emphasized by Baldwin and Clark (2003) and Narduzzo and Rossi (2004). What is important about modular architecture is that improvements (innovation) on one module do not trigger the need for changes in other modules of the product. This facilitates incremental innovation. Most noticeable OSS is available for free and fulfills the characteristics of a public good. In particular, the license terms – which we will discuss below – make the crucial difference between open source and commercial proprietary software, ensuring the non-exludability of OSS. Furthermore, the flow of information within and around OSS projects is extremely high and open to everyone. The Internet is the dominant tool facilitating this information flow. This close-to-costless information is a crucial characteristic of the open source development process. Information about new and ongoing projects is compiled on websites and in news groups. Bugs, bug fixes, module demands, etc., are listed and made available to programmers inside and outside the community. More importantly, there is a huge informational content in source code itself. The source code both enables other programmers to learn specific programming steps, and gives them access to understanding, criticizing and improving existing programming solutions. Also, empirical studies show that the 'average' OSS contributor is young and well educated. Hars and Ou (2002), for example, find 54 percent of the contributors in their sample to be less than 29 years of age and 72 percent have a college, masters or Ph.D. degree. Similar results are found by Hertel et al. (2003), Lakhani and Wolf (2005), Krishnamuturthy (2002), and Luthiger (2004). Finally, OSS programmers are often *user programmers*, meaning that they engage in a project because they are not satisfied with the existing software or simply because the required software feature does not exist. For example, Linus Torvalds needed a Unix for his PC, resulting in Linux (Torvalds and Diamond 2001), or Don Knuth needed a convenient tool for type-setting documents resulting in 'TEX'. Thus OSS programmers often benefit directly from developing and improving the software in question.

Obviously, not one OSS project is identical to any other. Nevertheless, we can identify some core agreements in the internal superstructure of successful OSS projects. The literature finds the following roles and responsibilities for the involved agents.

Roles and responsibilities in an OSS project

- *Users:* use the product of the project, report bugs, make feature requests.
- *Developers:* write code, produce documentation. All developers who contribute to a source file may add their name to the list of contributors.
- *Committers:* make frequent and valuable contributions to the project, earn voting rights when the project has to decide upon two alternative routes. All committers are named in the project credits.
- *Project Management Committee:* consists of committers who frequently make valuable contributions. Members of the project management committee are responsible for setting overall project direction.

Even though this presentation of roles and responsibilities hints at certain hierarchical structures within OSS projects, this is not how it is perceived within the communities. Community members view these structures as a natural extension of the division of labour. In particular, one does not achieve a place at one of the levels of these structures *ex officio*, but by contributing to the common OSS project. Important for the present issue of competition and innovation is the project management committee level. It is a disagreements at this level that can potentially lead to forking, and thus to intra-OSS competition. Yet the unifying force in the OSS project, both counterbalancing the threat of forking and generating the prime driver for individuals earning status within a project, is an alleged single dimension of quality. The claim within the community of programmers is that all developers and committers can and will be able to judge the quality of other programming work, precisely because the source code is open. It is argued that it is always possible to derive a clear ranking of competing modules, and most importantly, that there will typically be broad agreement in this ranking. Thus, in the terminology of the paper, an improvement of an ongoing OSS project must constitute an innovation step.

To see how these structures in the management of OSS projects interact with innovation, consider the case of Linux (see e.g. Torvalds 1999). Ever since Torvalds made the first Kernel freely available in 1991, Linux has rapidly developed into a stable and widely-used operating system. Two different innovation patterns can be observed in the development of Linux: on the one hand, the innovation pattern for the development of the kernel,

the crucial part of the operating system; and on the other, the development of other software closely connected to Linux such as hardware drivers or graphical engines.

In the case of the development of the kernel, the worldwide developed patches are collected and screened by a group of around six programmers working with Torvalds (the project management committee in the above terminology). As the kernel is crucial for the stability of the operating system, the programmers propose specific changes based on their tests. This selection od changes is passed on to Torvalds for final approval. These approved and stable running kernel versions are given an even version number.[7] This part of the innovation process therefore has a hierarchical pattern paired with singular technically determined quality criteria.

The development of additional modules (e.g. hardware drivers) and small applications, on the other hand, is more uncontrolled and undirected and takes place in LINUX newsletters, user groups, mailing lists and Internet pages. Projects in progress are presented on special information pages where concrete opportunities for co-operation are presented.[8] A selection of the patches, modules and applications offered is made by the community through prefered use of 'better'[9] versions. This part of the innovation process has a somewhat more anarchic pattern and user-determined quality criteria.

The existence of the two different innovation procedures within the development process of Linux is the result of the modular construction of the system. Although it seems likely that this was not foreseen by Torvalds at the start, this setting turns out to have the significant advantage that it promotes quick progress through the opportunity for independent development and debugging of different parts of Linux. In particular, the further development of the kernel and different modules can be carried out without any formal coordination of the different innovation activities.

2.1 The OSS licensing scheme

The cornerstone of the OSS movement is its ability to provide a coherent and incentive-compatible licensing scheme. In the software industry, two domi-

[7]Kernels which are in the process of being developed, patch levels have uneven version numbers. The current one is 2.7. These unstable so-called developer kernels are available on the Internet. See for example [www.kernel.org]. The current stable version is 2.6.10 released in December 2004.

[8]See e.g. [www.linux.org], [www.themes.org] or [www.linuxhelp.org].

[9]'Better' depends here on the tastes of the user. It could mean the reliability of the program as well as the innovativeness of the software or other characteristics of the software program.

nant licensing concepts can be distinguished: OSS and commercial software. In addition, there are less-relevant licensing concepts such as public domain software, freeware and shareware.

Commercial software is defined as being software which, through certain licensing conditions, prohibits or limits usage, modification and duplication. Modification is often not an option as the source code is not available. The possibilities for usage are limited, as the source code written by the programmer is replaced by a compiled code. In contrast to commercial software, the source code of OSS is freely available. The basic idea and main aim of open source (or free software[10] as it is sometimes called) is free usage and the possibility for further development by the user. 'Free', in this context, means that the user is free to duplicate, modify, and distribute OSS. The term open source software covers a number of varying forms of software licenses, which all seek to ensure free access for the user.

Key features of OSS licenses

- *Free distribution:* No licensing fees may be charged for the software.
- *Source code available:* The program must contain the source code or at least indicate an alternative method of receiving it, i.e. via the Internet.
- *Derived works must be published as OSS:* Any modification of the program falls into the same licence as the original software.

One of the most popular OSS licences is the GNU-GPL which is a free software and copyleft license written by the FSF in 1989.[11] Linux, for example, is subject to the GNU-GPL. The most important condition of the GNU-GPL apart from the above features is the 'No guarantee' feature. Unless agreed otherwise, there is no guarantee for the software. The risks in the

[10] The terms *free software* (introduced by the FSF) and *open source software* (introduced in 1998 by a group of programmers on the open source meeting in Palo Alto, California) are nearly interchangeable. Arguably, the term open source software holds the danger that the aspect of the free software concept, namely the free distribution and modification of the software, is not sufficiently emphasised. Therefore the concept of open source is declined by the FSF. In this article both concepts are used synonymously.

[11] The exact wording of the GNU GPL can be found at [www.linux.de/linx/gnu.html] or the pages of the Free Software Foundation at [www.fsf.org/copyleft/gpl.html]. In addition to the GNU GPL there are many other OSS licences. Some of the more important are: BSD License (original); BSD License (modified); GNU - Library General Public License (LGPL); X11 License; Apache License; Mozilla Public License (MPL); Netscape Public License (NPL); Qt Public License (QPL); Artistic License.

level of quality and productivity are under the principle of caveat emptor.

Similarly to open source software, *Public Domain* software also makes its code freely available. The author allows modifications and duplications without receiving any reimbursement, thereby giving up any copyright or influence on his work. Representatives of the Free Software Foundation (FSF) and the Open Source Movement have spoken out against the concept of public domain software, as a user can make changes to a program, turn it into commercial software and thereby withhold it from the public. Thus the central difference between public domain software and open source software is that the latter is protected from withholding and absorption by commercial software by certain licensing conditions. As in the case of public domain software and OSS, *freeware* and *shareware* products can also be duplicated. Freeware and shareware allow the user to duplicate and distribute the software, but not to modify it. In this respect, the source code of freeware and shareware is just as inaccessible as it is in the case of commercial software. In order for potential buyers to investigate a piece of software, shareware products have limited usage which is generally controlled by a fixed period of evaluation or the amount of usage itself. Accordingly it is more of a marketing concept than a licensing concept. Fees are charged if the shareware is copied or distributed, which is a contrast to OSS, and to public domain software and freeware, which can be obtained over the Internet free of charge. It is important to note that with commercial software, companies generally make most of their profits in licensing and duplications fees. Table 1 summarises the characteristics of the various licensing concepts.

Table 1: Comparison of different licensing concepts

Software Type	Availability of source code	Possibility of duplication	Possibility of modification	Free of charge
Commercial	no	no	no	no
Freeware	no	yes	no	yes
Shareware	no	limited	no	limited
Public Domain	yes	yes	yes	yes
Open Source/ Free	yes	yes	yes	yes (possibility of fees)

2.2 Stylised characteristics of the innovation process

The organizational and motivational features and the ownership rights and licensing conditions introduced above have an important influence on the OSS development process. Abstracting from the details of single OSS projects, there are a number of stylized characteristics of the OSS innovation process valid for the majority of OSS projects. The innovation process of OSS has several pro-innovative features compared with the innovation procedures of commercial enterprises.

Pro-innovative features of OSS

- *Army of programmers:* huge number of programmers, open communities, support functions, collective problem awareness
- *High knowledge spillovers:* the source code contains both the product and the information of how the product is constructed, huge opportunities for learning and knowledge spillovers
- *Motivation effect:* high motivation of programmers, gift culture, intrinsic motivation, peer group monitoring
- *Boundless cooperation:* absence of commercial conflicts, open innovation system, modular product architecture
- *User-developer effect:* User developers imply detailed knowledge of what improvements are needed, innovation close to the customer.
- *Disciplining forking-thread:* constant risk of forking and branching of the project if it does not life up to community expectations/standards.
- *Zero-costs effect:* no R&D costs, no sunk costs, no research investments that must be recovered, willingness to abandon development dead-ends.

The huge *number of programmers* is partly a result of the fact that the source code is open and available for free. For example, universities often use OSS products in their classes for programming training because of the availability of the code. This of course adds a substantial amount of contributions to the development of the selected OSS projects. Furthermore, young programmers get in touch with the OSS community (Dempsey et. al., 1999). The key pro-innovative channel of engaging a huge number of programmers

is simply that more programmers improve, bug-check and bug-fix, and further develop OSS products than in commercial enterprises. The operating system Linux, for example, is estimated to be maintained and developed by some 10,000 active programmers. In commercial enterprises the number of software programmers is lower and they have to cater to a larger number of software projects. Linux's competitor, Microsoft, for example, which offers around 5,500 different software solutions accompanying their well-known operating system has a total of approximately 15,000 engaged in research & development activity.[12]

An important difference from closed source software is the ability of OSS to facilitate *knowledge spillovers* within the community. Given enough time, a computer programmer can, from available source code, deduce, learn, and improve the programming steps developed by other programmers. Thus programmers can read, understand, and learn from the programming innovations of other programmers. Thus the knowledge diffusion within OSS projects is maximized (cf. Raymond 2000b). Closed source 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. 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 a compiled piece of software, which disguises cumbersome or faulty programming steps.

This leads to another striking feature of OSS production, namely the *high motivation* of the contributing programmers. First of all by the very nature of voluntarism, programmers usually only work on projects that they enjoy working on. Another reason for high motivation is signaling as the result of one's programming work being published, and the authors' name stated. The providers of the OSS benefit in some ways from being able to signal their programming skills; either through improving prospects on the job market, Lerner and Tirol (2002) and Raymond (2000b), and/or enhancing reputations within the community of programmers (see Raymond 2000a, and Torvalds and Diamond 2001). Such signals are institutionalised in OSS activity: an OSS license obliges its programmers/users to document all program changes made (including naming the author of the changes) in the software itself. Thus, each programmer is interested in maximizing the signal value of his work by providing high quality software. Finally, the important side effect is that peer group monitoring – mentioned above – also heightens motivation:

[12]See [www.microsoft.com/presspass/fastfacts.htm] and [www.microsoft.com/Germany/] and SuSE, C't and Linux-Magazin.

precisely because the source code is open, certain programming steps and the quality and elegance of the programming work is visible to every other programmer – as opposed to compiled software, where the design and quality of the programming is not disclosed directly – and this acts as a motivational driver to deliver better programming work.

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 emerge. This results in two factors making that make the OSS innovation process pro-innovative. First, and as mentioned above, the unlimited access to the source code leads to a very high knowledge diffusion. Second, as no commercial interests prevent the co-operation between programmers, beneficial combinations of complementary programming skills can be exploited. It is also noticeable that the voluntary and innovative co-operation of developers allows for individual and specific solutions for software problems and programming queries to be found. A similar spectrum of solutions provided by commercial software is unthinkable because of the cost involved. Finally, under the heading of boundless cooperation, also the innovation advantages stemming from a modular product architecture must be captured. Improvements (innovation) on one module do not trigger change requirements in other modules of the product, thus rapid innovation is facilitated (Baldwin and Clark, 2003; Narduzzo and Rossi,2004).

The fifth important advantage, in particular in comparison to commercial software development, is the close connection between users and developers. Often user and developer are the same person as the need for a particular solution is a driving motive to start an OSS project or to develop a certain extension to an ongoing OSS. This *user-developer* element is taken to constitute a major advantage of the OSS development process, see Kuan (2001), Franke and von Hippel (2003).[13] But even if the programmer and the user are two different people, the communication between them takes place very directly because the contributor of OSS is published and can be addressed directly. Thus, the communication is not hampered by sales and support departments or other intermediaries. These close connections enable much quicker problem-solving and removal of bugs in the software.

The pro-innovative impact of the *forking-thread* effect has not been discussed in the literature thus far. Forking and branching describes the split-

[13]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 analyze cases of user communities that support innovation activities for sports-related products.

ting up of OSS projects into rival and competing development streams. The right to modify and distribute as OSS a splintered version of an existing 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 the future development is thus based on democratic principles, where the majority of the most important contributors usually decide. This has two important effects on the OSS innovation process. First, it ensures that technological aspects are central to the decision in which direction the further development of an OSS project should go. Second, new innovations are implemented immediately.[14]

Finally, the *zero cost effect* results from the fact that the development of an OSS is not restricted by cost considerations. Thus, in the decision between two technological alternatives, costs are not a decisive criterion. Therefore technologically superior but not economically profitable software solutions can be realised in an OSS environment. The same holds true in the case of further development of software. As an update of software is connected to development and release costs, commercial enterprises are tentative about releasing minor software updates. OSS communities, on the other hand, act following the principle of 'release early, release often' (cf. Ramond 2000b, Chapter 4). Thus, the disrespect for sunk costs makes OSS products more prone to changes and improvements.

Beside these advantages in the innovation process of OSS, there are, of course, also severe disadvantages in particular in comparison to commercial development of software, several of which we have already mentioned. The key problems can be summarized as follows.

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

Anti-innovative features of OSS

- *Development redundance:* redundant and abandoned developments, double efforts
- *Undirected process:* undirected development process, missing items problem, incomplete innovation
- *Unhealthy forking:* splitting of the code base, incompatibility problems, fragmentation of network effects
- *Supply side constraint:* splitting the combined available supply of programming skills across to many projects
- *Code reuse problem:* extensive code reuse, difficult to track down subsequent changes/improvements

Due to the freedom within the various OSS communities, there are a huge number of *redundant developments* which often result in the 'reinvention of the wheel'. From an economic point of view, this is a waste of resources. Arguably, it may be practical and feasible for OSS projects to search the space of conceivable software solutions through the power of the size of the community (army of programmers), thus simply running an immense amount of trial and error experiments. However, the economic and innovative efficiency of this approach is questionable. This leads directly to the next problem, namely that of undirected development.

The problem of an *undirected innovation* process is problematic in particular for professional users. 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 driver or getting it programmed for pay – on the decision of programmers to develop the driver. Furthermore, because of the absence of a liable coordinating institution, it can neither be guaranteed that existing hardware or software is supported by the operating system in later versions nor that newly purchased hard- 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 with the whole community through the publication of the source code. In contrast, commercial software enterprises usually pay much attention to the backward and forward compatibility of their software, as well as to the support of older and newly emerging hardware.

Next we come to the *unhealthy forking* issue mentioned in the quotation presented in the introduction and the issue of potentially hitting a *supply side constraint*. Once forking has occurred, it is accompanied by several anti-innovative effects. First, the appearance of 'forks' ultimately reduce the number of programmers working on each of the forks, thus reducing the rate of innovation via a supply-side effect. Second, forking could lead to incompatible standards reducing existing network effects. Overall, the potential supply side restriction really is the flip side of the army of programmers.

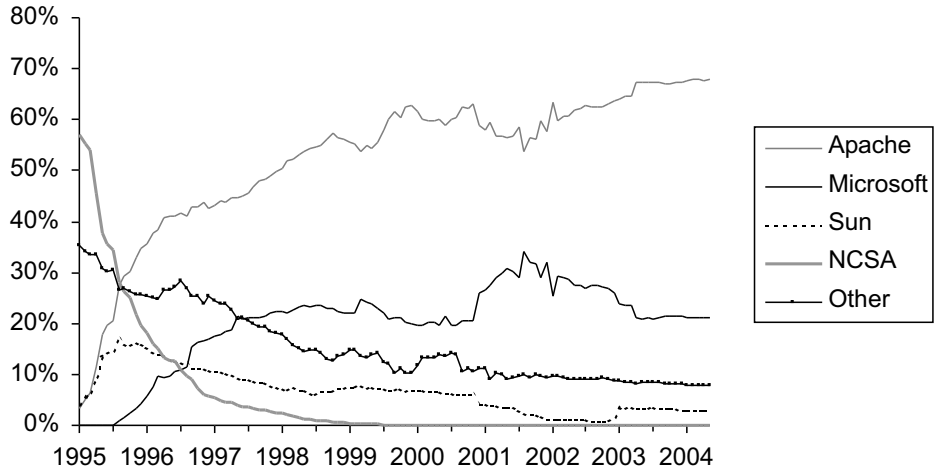
Finally, as discussed by, e.g., Spinellis and Szyperski (2004) the extensive *code reuse* embedded in the production process of OSS may, to some extent, create a barrier to innovation. Everything from a few lines of code to entire program structures may be and are reused in OSS projects. This really constitutes the flip-side of the knowledge spillover effect discussed above. The problem of code reuse, however, is that different versions of the same programming item may exist at the same time, and once reused in other parts of the same or even another project, an improvement of the initial part is close to impossible to trace onwards to all the subsequent places of use. Furthermore, Spinellis and Szyperski (2004) also argue that due to code reuse, shoddy and substandard code may be duplicated and spread, since the monitoring practices and quality standards of different OSS projects diverge widely.

Thus, in concluding this section, it must be noted that from the above considerations, one cannot judge if the OSS development process is more or less efficient than the commercial model. In order to answer the question one needs to assess the costs of the OSS process, which, even though programmers are unpaid, undoubtedly exist in the form of time and effort spent and opportunity costs in general. To be clear: we are uncertain whether the commercial development process if given the same inputs – that is the same army of programmers and degree of dedication and motivation – might not result in *more* innovation. Arguably, commercial software production may be unable to generate the same motivational drivers as OSS, but this is not the same thing as saying the OSS is a more efficient production method. Nevertheless, what can be deduced from the above discussion is that the occurrence of competition caused by the entry of OSS (extra OSS competition) or through forking (intra-OSS competition) will most likely have an impact on individual and accumulated innovation activity in the software sector. It is this effect that we attempt to shed light upon in the next section by presenting empirical indicators of innovation activity in the software industry following the entry of OSS.

3 Innovation impact of OSS: empirical indicators

The impressive success of the OSS development model is best illustrated by the rapid capture of market shares in traditional commercial software markets by OSS competitors. For example, in the lucrative server operating systems market the market leaders of the mid-90s (Novell and Unix) have lost market shares over the past 10 years, two entrants – one commercial (Microsoft with Windows NT) and one OSS (Linux) – now dominate the market with approximately 40% market share each (see Bitzer, 2004). The same has happened in the market for web servers (see Figure 1). Former market leaders like the National Center for Supercomputing Applications (NCSA) and Netscape (included in SUN in Figure 1) have lost their dominating market position against the commercial entrant Microsoft and the OSS Apache.

Figure 1: Market Shares in the Market for Web Servers



Source: [www.netcraft.com].

Although commercial software enterprises might lose market shares with the entry of OSS, this is of course not evidence of either a pro- or anti-innovative effect in this market segment. Not only is there competition from OSS, but also from commercial software enterprises that have been able to enter the software markets discussed successfully. What is needed for such an analysis is data on the technological innovations introduced. Data on technological progress in software technologies is, however, difficult to compile. We use here the following proxi. We regard the release history as a possible indicator of technological progress in software technology. This is

because the release of a new version creates costs, not only for the software firm or OSS developer but also for the customers implementing it (switching costs). A customer is only willing to introduce a new software version if the benefits thereof are greater than the implementation costs. Thus, the resulting release costs will only be justified if there is at least a certain technological improvement in newly released software.[15] Furthermore, it can be expected that the higher the release costs, the higher the technological improvement necessary to offset investment. In particular, it seems that the rate of commercial software and OSS innovation has increased with emerging competition. The release history of Microsoft's Windows (see Table 2) shows that the intervals between major releases have decreased continuously since the beginning of the 1990s.

Table 2: Release history of MS Windows[16]

Release date	Version
1985	Windows 1.0
1987	Windows 2.0
1990	Windows 3.0
1995	Windows 95
1998	Windows 98
2000	Windows ME
2001	Windows XP

Source: Information published at: [www.microsoft.com].

Compare this to some key indicators of innovative activity at OSS competitor Linux (Table 3).

[15] As was discussed under the zero cost effect in the previous section, the release cost of OSS is of course only a fraction of that of a commercial software producer. Hence it is important to not directly compare release frequencies across software producers but only the changes in release frequencies of individual producers across time and in reaction to entry and competition.

[16] Further minor releases have been Windows 3.1 and Windows for Workgroups 3.11 in 1993 and Windows 98 SE in 1999.

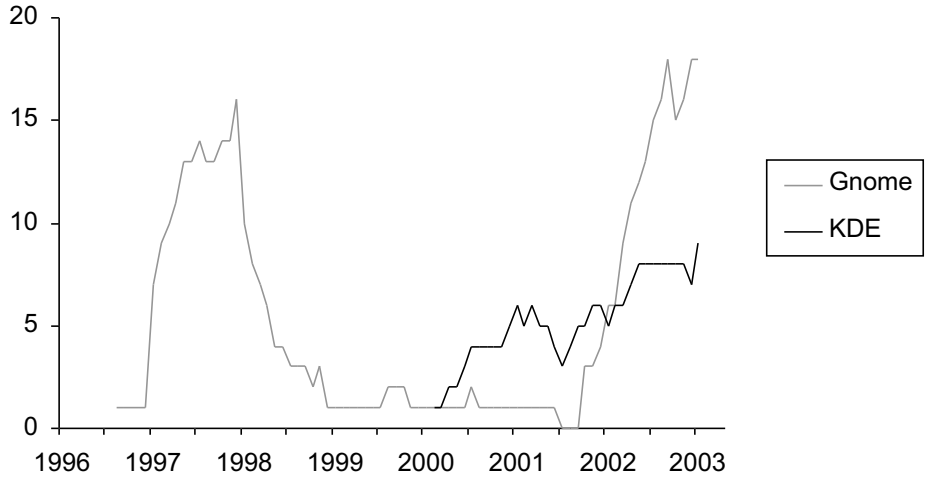
Table 3: Development of Linux 1991-2004

Year	Version	Users	Lines of code
1991	0.01	1	10,000
1991	0.10	100	18,000
1992	0.96	1,000	40,000
1993	0.99	20,000	80,000
1994	1.0	100,000	180,000
1995	1.2.0	500,000	310,000
1996	2.0.0	1,500,000	780,000
1997	2.0.2x	3,500,000	800,000
1998	2.0.3x	7,500,000	1,500,000
1999	2.2.0	12,000,000	1,800,000
2000	2.4.0	18,000,000	3,380,000
2002	2.5.37	23,400,000	5,100,000
2003	2.6.0	30,420,000	6,000,000

Source: McHugh (1998), information from SuSE, author's enquiries. E.g. information on [http://www.win.tue.nl/ aeb/linux/lk/lk-1.html].

A similar trend of increased innovation activity is also found for competition among OSS projects, as documented in Figure 2. The case of the two graphical OSS desktop environment competitors GNOME and KDE resembles the features of an OSS fork. The data does not indicate any slowdown in innovation subsequent to increased competition (the entry of KDE). On the contrary, both branches appear to have gained momentum once competition kicked in (see Figure 2).

Figure 2: Number of Version Releases in the Past 12 Month



Source: www.kde.org , www.gnome.org.

Quite the reverse can be observed in the case of decreasing competition, e.g. in the case of the Microsoft Internet Explorer (see Table 4). In the early years 1995-1998 when Microsoft was facing competition from Netscape's web browser, the period between major releases decreased steadily. After 1998, however, when Microsoft won the battle and dominated the web browser market, the delay between major releases appears to have grown. The next release, Internet Explorer 7.0, is expected for late 2005, about four years after the last major release 6.0.[17] Thus by this measure, a decrease in competition (the exit of the Netscape browser) appears to be associated with a slowdown in innovation.

Table 4: Release History Internet Explorer 1995-2005

Version	Date
1.0	Aug 1995
1.5	
2.0	Nov 1995
2.01	
2.1	
2.5	
3.0	Aug 1996
3.01	Oct 1996
3.02	
3.03	
4.0	Oct 1997
4.01	Nov 1997
5.0	Sept 1998
5.01	
5.5	Jul 2000
6.0	Oct 2001
6.0 SP1	Sept 2002

Sources: http://www.microsoft.com/windows/WinHistoryIE.mspx and http://www.blooberry.com/indexdot/history/ie.htm and http://en.wikipedia.org/wiki/Microsoft_Internet_Explorer.

However, recently – and conceivably in reaction to this slowdown – a new competitor has successfully entered the market: 'Mozilla Firefox' a stand-alone OSS web browser. Mozilla Firefox's version 1.0 released in November 2004 was downloaded 5.6 million times within the first two weeks.[18] Furthermore, experts report that Microsoft's Internet Explorer has lost 5 percent

[17]The Internet Explorer 7.0 is planned to be released with Windows Longhorn the successor of Windows XP.

[18]See information on [http://www.mozilla.org/news.html].

of market share to Mozillas Firefox over a six-month period since May 2004 (see Festa 2004). Table 5 documents the impressive development efforts of the Mozilla Firefox programming community.

Table 5: Release History Firefox 2002-2005

Version	Date
0.1 *	23.09.02
0.2 *	01.10.02
0.3 *	16.10.02
0.4 *	30.10.02
0.5 *	07.12.02
0.6 **	16.05.03
0.6.1 **	28.07.03
0.7 **	15.10.03
0.7.1 **	26.10.03
0.8	09.02.04
0.9rc	09.06.04
0.9	15.06.04
0.9.1	28.06.04
0.9.2	08.07.04
0.9.3	04.08.04
0.10rc	11.09.04
0.10	14.09.04
0.10.1	01.10.04
1.0rc1	27.09.04
1.0rc2	04.11.04
1.0	08.11.04

Remarks: * codename "phoenix"; ** codename "firebird"
Source: www.mozilla.org.

The above evidence on innovation activity following the entry an emergence of OSS can only be a first attempt at answering the issue of the innovation impact from intra- and extra-OSS competition. The data and cases presented do not offer evidence of an anti-innovative impact; on the contrary, if anything, the entry of OSS into commercial segments of the market or forking in ongoing OSS projects appears to be associated with increased innovation activity.

4 A model of Software Competition

4.1 Intra- and extra-OSS competition

Competition between a profit oriented commercial 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.[19] Third, the behavior of the market participants is strongly influenced by an exogenous technological factor.

First, the incentives of OSS developers have recieved a great deal of attention. 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, 2000b; Torvalds and Diamond, 2001; Lerner and Tirole, 2002). Hence, even though a profit motive can be ruled out for OSS developers, they nevertheless maximise these other payoffs. Although the incentives of the two types of software producers are different, both incentives are positively correlated with the dissemination of their respective software product. While commercial firms are interested in increasing their profits by benefitting 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 signaling value (Lerner and Tirole, 2002).[20] Thus, commercial firms and OSS developers can still be assumed to maximise their respective payoffs, 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 commercial software producers and OSS developers.

Second, 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.[21] Since there can be no talk of either quantity *or* price competition, our paper starts out by formulating an (admittedly un-

[19] 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).

[20] Furthermore, the presence of network effects in software use could explain why both types of software producers value dissemination.

[21] In fact also the price of most commercial 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.

conventional) 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. Thus, the technology of a software includes the entire bundle of its technological characteristics.

These considerations lead us directly to the third 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. 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 the current point in time.[22] On the other hand, the global technological level itself is constantly changing. It is set by the developments in the globally available technology which influences 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, i.e. innovation.

4.2 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. 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.

[22] 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.

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 or changes in consumer demand.

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, and on the interdependence between the two market segments, i.e. the two products are imperfect substitutes. 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

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

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

Parameter $\beta > 0$ is a measure of the substitutability of the two products and α is a positive constant. The expressions $\frac{\tau_t^i}{T_t}$ 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. Equations (1) and (2) also include the effect of external technological development in T_t which devalues the real technology level and dissemination of both software products.[23]

Even though the model avoids the notion of price or quantity competition, the principles of maximisation are still applicable. Assume that firms derive some payoff from each distributed/sold unit of software. In particular there is a gain ρ, which could represent the reputational or signaling value for an OSS producer or more conventionally the per unit monetary reward for a proprietary software firm. There is also a cost C, 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, i.e. 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 $C = c\frac{\tau_t^i}{T_t}$ the gain function for firm i can be stated as:

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

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

How does this 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, realises that $\tau_t^{Ma} = \tau_t^{Mb}$. Rewriting the monopolists gain function for market i after setting in (1) and (2) respectively, gives

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

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

4.3 Innovation – setting technology levels

In the case of a software duopoly, where firms behave noncooperatively and simultaneously have to choose their respective technology levels to maximise (3), the first order condition for firm a after substituting in q_t^a from (1) becomes

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

and similarly for firm b.

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

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

Lemma 1. *The technology level set by a software duopoly increases for a higher payoff, ρ, a falling cost, c, and a higher degree of substitutability, β.*

Thus, 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 software duopoly will produce a higher technological level. Similarly a duopoly with a higher payoff 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 the software duopoly triggers firms to set a higher technology level.

Compare this to the asymmetric case. In a software duopoly with heterogeneous costs, c_a and c_b, firm a's Nash equilibrium technology level would become $\tau_t^a = \frac{\rho T_t(\alpha+2\beta)(c_a\beta+c_b(\alpha+2\beta))}{2c_ac_b(\alpha 2+4\alpha\beta+3\beta 2)}$, such that $\frac{\partial \tau_t^a}{\partial c_a} < 0$ and $\frac{\partial \tau_t^a}{\partial c_b} < 0$. A low cost competitor b increases the technology level set by firm a. Thus, we obtain:

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

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. Return now to the assumption that $c_a = c_b = c$, such that the impact from OSS on the for-profit competitor stems solely from the strategic interaction of the two firms rather than from the possible cost advantages that an OSS competitor may have.

Compare the above findings to a software monopolist maximising (4), and expecting all future technology level T_{t+j} , $j = (1, 2, \dots)$ to be equal to T_t. Such monopolist has first order conditions for market segment A and B that define the optimal technology level as

$$\tau_t^{Ma} = \tau_t^{Mb} = \frac{\rho T_t}{2c}. \tag{7}$$

Lemma 3. *The technology level set by a software monopolist increases for higher payoff, ρ, and a falling cost, c, 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, an OSS monopolist will provide a higher technology level than a for-profit monopolist. Yet, if the for-profit monopolist has a the higher payoff ρ compared to the payoff for the OSS developer, than the reverse conclusion applies.

Comparing (6) to (7) leads to the following 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}.*

Proposition 1 states that a monopolist will choose a lower technology level than a software duopoly. 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. This beneficial impact of entry occurs no matter if the entry takes place in a commercial software market or in an existing OSS market, i.e. 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 optimise their technology position with respect to the outside technological development, they also innovate in anticipation of the other firm's innovation activity. This effect matters more the closer substitutes the goods are, i.e. the closer competitors the two firms are.

Finally, from the above results we would expect to observe a higher frequency of technological adjustments (new releases) for OSS markets and for software markets that experience entry or forking. Thus such markets will be tracking the technological development more closely, i.e. adjusting to smaller shocks, while the monopolist stands still longer, etc. The model further suggests that in mixed markets where one competitor is a commercial software enterprise and the other is an OSS developer, the chosen technological level and the rate of innovation are higher than in a pure commercial software duopoly, but lower than in a pure OSS duopoly. Thus, within the present framework, evaluated in terms of the technology level and progress, the pure OSS duopoly dominates all other market structures treated in the paper.

Furthermore, assuming comprehensively that a higher technological level and a faster adjustment to global technological developments are associated with higher user utility, and given that increasing the number of software firms meets no supply constraint in terms of programmer capacity, then our model implies that users benefit from entry of an OSS developer into a software market dominated by a monopolist. This holds for both cases: whether the monopolist is a commercial software enterprise or an OSS developer, as well as for the case of forking and branching in OSS projects.

5 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 operating system market. Some observers fear that the progress in software technology, that is the rate of innovation, will slow or even stop altogether as a consequence of OSS entry into former highly concentrated (monopolistic)

markets. Possible channels for such anti-innovative impact are the reduced profitability and thus resulting lower R&D expenditures in the industry or an allegedly lower efficiency of the OSS innovation process. The paper reviews the OSS innovation process and highlights the pro- and anti-innovative features of this development model. The most important pro-innovative features of OSS development are based on the huge number of programmers involved, and the high knowledge diffusion and motivation among programmers. Anti- innovative OSS features, on the other hand, are the potentially high development redundance and the problems associated with an undirected innovation process as well as the occurrence of forks and branches in OSS projects. Next, the paper presents empirical evidence on innovation in the software business post OSS. The data collected does not reveal an anti-innovative effect. On the contrary, it suggests that, if anything, increasing competition in the software industry promotes innovation.

Based on these observations, 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 a pure OSS duopoly dominates in terms of technology levels and rate of innovation monopolies (either commercial 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 is knowledge-intensive, that is software.

To sum up the specific features of the OSS innovation process identified in this paper, much of the empirical data available to date and our theoretical considerations on the effects of competition in a technology setting software duopoly all suggest a pro-innovative effect of the emergence of OSS.

Finally, the findings of the present paper give rise to a wide range of new and fundamental questions for understanding the software business post OSS. From all we know about OSS and its successful entry into the software industry, one would judge that it has come to stay. OSS is able to challenge and potentially replace incumbent commercial firms. One key towards understanding this success is – in our view – the impressive innovative performance of the OSS development model. However, only if paired with simple and accessible distribution concepts and high levels of user-friendliness that are acceptable for a wide range of users other than specialized user programmers

can this potential be unrevealed. Furthermore, the OSS development model might well extend to other business applications, and it has certainly always featured as a model of innovation in academia, where often the findings and methods of individual researchers and groups of researchers are made openly accessible through publication in journals and presentation at conferences. In fact, these principles have been the backbone of scientific progress for centuries. Viewed as an extension of these fundamental principles of scientific research, OSS might not be such a new phenomenon after all, and rather the brief heyday of closed source software in in the 1980s and 90s would have been an exception.

References

Arrow, Kenneth J. (1962): Economic Welfare and the Allocation of Resources for Invention, in R. R. Nelson (ed.): T*he Rate and Direction of Inventive Activities*, Princeton University Press, pp. 609-625.

Baldwin, Carliss Y., and Kim B. Clark (2003): The Architecture of Cooperation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model?, *Harvard Business School Working Paper Series*, No. 03-209.

Barcomb, Ann (2004): Open Source and NASA's Mars Rover, in: ONLamp.com, [http://www.onlamp.com/lpt/a/5053], February 2, 2004.

Bitzer, Jürgen (2004): Commercial versus open source software: the role of product heterogeneity in competition, *Economic Systems*, Vol. 28, No.4, pp. 369-381.

Bitzer, Jürgen, Wolfram Schrettl and Philipp J.H. Schröder (2004): Intrinsic Motivation in Open Source Software Development, Discussion Paper, No. 2004/19, Department of Economics, Free University Berlin.

Bitzer, Jürgen and Philipp J.H. Schröder (2005): Bug-Fixing and Code-Writing: The Private Provision of Open Source Software, *Information Economics and Policy*, , Vol. 17 (3), pp. 389–406.

Bitzer, Jürgen and Philipp J.H. Schröder (2006): The Economics of Open Source Software Development: An Introduction, in: Bitzer, J. and P.J.H. Schröder (eds.) The Economics of Open Source Software Development, Elsevier Science Publishers.

Bresnahan, Timothy F. and Shane Greestein (1999): Technological Competition and the Structure of the Computer Industry, *Journal of Industrial Economics*, Vol. 47, No. 1, pp. 1-40.

Casadesus-Masanell, Ramon and Pankaj Ghemawat (2003): Linux vs. Windows: Modelling Competition between Open-Source and Closed Software, mimeo.

Dempsey, Bert J.; Weiss, Debra, Jones, Paul; Greenberg, Jane (1999): A Quantitative Profile of a Community of Open Source Linux Developers, SILS Technical Report TR-1999-05.

Feller, J. and B. Fitzgerald (2002): *Understanding Open Source Software Development*, Amsterdam: Addison Wesley Longman.

Festa, Paul (2004): Firefox cutting into IE's lead, in: CNET News.com, [http://news.com.com/Firefox+cutting+into+IEs+lead/2100-1025_3-5463513.html], November 22, 2004.

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

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

Hars, A. and S. Ou (2002): Working for Free? Motivations for Participating in Open-Source Projects, *International Journal of Electronic Commerce*, Vol. 6 (3), pp. 25-40.

Hertel, Guido, Sven Nieder and Stefanie Herrmann (2003): Motivation of Software Developers in Open Source Projects: An Internet-based Survey of Contributors to the Linux Kernel, *Research Policy*, Vol. 32(7), Special Issue: Open Source Software Development, pp. 1159-1177.

Johnson, J. P. (2001): Open Source Software: Private Provision of a Public Good, *Journal of Economics and Management Strategy*, Vol. 11 (4), pp. 637-662.

Krishnamurthy, Sandeep (2002): Cave or Community? An Empirical Examination of 100 Mature Open Source Projects, in: *First Monday*, Vol. 7, No. 6, [www.firstmonday.org].

Kuan, J., (2001): Open Source Software as Consumer Integration into Production, SSRN Working Paper, [http://ssrn.com/abstract=259648], date 29 July 2004.

Lakhani, Karim R. and Robert G. Wolf (2005): Why Hackers Do What They Do: Understanding Motivation Effort in Free/Open Source Software Projects, in: Feller, J., Fitzgerald, B., Hissam, S. and Lakhani, K. R. (eds.) Perspectives on Free and Open Source Software. MIT Press, Cambridge, pp. 3-21.

Lerner, J. and J. Tirole (2002): Some Simple Economics of Open Source, *Journal of Industrial Economics*, Vol. 50 (2), pp. 197-234.

Luthiger, Benno (2004): Fun and Software Development, in: FASD Study: Provisional Results, University Zürich.

Microsoft (2001): Microsoft PressPass Information for Journalists, http://www.microsoft.com/presspass, download date 11.04.2003.

Mustonen, Mikko (2003): Copyleft – The Economics of Linux and other Open Source Software, *Information Economics and Policy*, Vol. 15 (1), pp. 99-121.

Narduzzo, A. and A. Rossi (2004): The Role of Modularity in Free/Open Source Software Development, in: Koch S. (ed.): *Free/Open Source*

Software Development, Hershey, Pa: Idea group, pp. 84-102.

O'Mahony, Siobhan, and Fabrizio Ferraro (2004): Managing the Boundary of an 'Open' Project, in: *Market Emergence and Transformation*, edited by John Padgett and Walter Powell, forthcoming.

Raymond, E. S. (2000a): Homesteading the Noosphere, Revision 1.22, 2000/08/24, first version 1998, [http://www.catb.org/ esr/writings/homesteading/homesteading/].

Raymond, E. S. (2000b): The Cathedral and the Bazaar, Revision 1.51, 2000/08/24, first version 1997, [http://www.catb.org/ esr/writings/cathedral-bazaar/cathedral-bazaar/].

Rosenberg, D. K. (2000): *Open Source: The Unauthorized White Papers*, B&T; IDG Books Worldwide.

Rossi, M. A. (2006): Decoding the Free/Open Source Software Puzzle: a survey of theoretical and empirical contributions,in: Bitzer, J. and P.J.H. Schröder (eds.) The Economics of Open Source Software Development, Elsevier Science Publishers.

Schumpeter, Joseph A. (1934): *Theory of Economic Development.* Cambridge, Mass.: Harvard University Press.

Schumpeter, Joseph A. (1942): *Capitalism, Socialism and Democracy.* (reprint), HarperPerennial; New York.

Shah, S. K. (2003): Understanding the Nature of Participation & Coordination in Open Source Software Development Communities, Dissertation, Chapter 4, MIT Sloan School of Management.

Spinellis, Diomidis and Clemens Szyperski (2004): How is Open Source Affecting Software Development?, *IEEE Software*, No. 1, pp.28-34.

Stallman, R. (1999): The GNU Operating System and the Free Software Movement, in: *Open Sources: Voices from the Open Source Revolution*, Chris DiBona, Sam Ockman, and Mark Stone (eds.), O'Reilly & Associates: Sebastopol, CA, pp. 53-70.

Torvalds, L. (1999): The Linux Edge, in: C. DiBona, S. Ockman and M. Stone, eds., *Open Sources: Voices from the Open Source Revolution*, O'Reilly & Associates, CA, pp. 101-111.

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

Ulhøi, John P. (2004): Open Source Development: a Hybrid in Innovation and Management Theory, *Management Decision*, Vol. 42, No. 9, pp. 1095-1114.

Ulrich, K. (1995): The role of product architecture in the manufacturing firm, *Research Policy*, Vol. 24 (3), pp. 419-440.

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

Handelshøjskolen i Århus

Aarhus
School of Business

ISBN 87-7882-106-1 (online)

Department of Management and International Business

Aarhus School of Business
Fuglesangs Allé 4
DK-8210 Aarhus V - Denmark

Tel. +45 89 48 66 88
Fax +45 89 48 61 25

www.asb.dk