A Survey of Open Source Software

Sameer Alam[1], Md Saqlain Salim[2], Md Shahabuddin[3], Md Zishan[4], Rittik Maity[5], Ravi Kant Ray[6], Shanta Phani[7]

[1,2,3,4,5,6,7]*Department of Computer Science and Engineering,Bengal Institute of Technology, Kolkata, West Bengal*

Abstract - The entry and success of Open Source Software (OSS), e.g. 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 open-source software innovation and highlight the impact of increased competition and different cost structures on innovative activity in the industry. 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 captures 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.

Keywords - economics of open source, innovation, open-source software, software competition, strategic interaction, software market.

1. INTRODUCTION

The striking difference between Open-Source Software (OSS) and Proprietary Software (PS) 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 the participation of user-programmers, high knowledge spillovers, contributions by volunteers, and reuse of source code.

The question of what impact this unusual development method has on innovation activity in the software sector and how it competes with its paid counterparts has received surprisingly little attention thus far.

Existing research that is in part related to this fundamental question is that of Casadesus-Masanell and Ghemawat (2006) and Bitzer (2004), who analyzed 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 is those of Economides and Kat samakas (2006a) and Bitzer and Schr¨oder (2006). Economides and Kat samakas (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 number of user-programmers, and the total cost of ownership (TCO). Bitzer and Schroder (2006) review the 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 difference 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 Schroder (2006). 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, i.e. 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.

We examine the impact of increased competition on innovation activity in two ways: first, we analyze 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 analyze 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. Casadesus-Masanell and Ghemawat (2006), Economides and Kat samakas (2006b), Bitzer (2004)). However, departing from preceding studies, we do not assume a no-cost development of OSS. Rather, by analyzing 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 the 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 payoff 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 counterintuitive 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, i.e. 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 counter-balance a higher TCO with more advanced technological content in order to maintain a satisfactory level of product dissemination.

The remainder of the paper proceeds as follows. The following section discusses the 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 setup to analyze the entry of OSS into formerly monopolistic markets, and Casadesus-Masanell and Ghemawat (2006) analyzed 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. Bitzer et al. 2004, Luthiger 2005). 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 (Lepp¨am¨aki and Mustonen 2004a, 2004b, Lerner and Tirol 2002, and Raymond 2000b), and/or through enhanced reputation within the community of programmers (see Raymond 2000a, and 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 programmer's 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, 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 (Haefliger et al. 2006, Lakhani and von Hippel 2003).

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.

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 licenses (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, users and developers 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). But even if the programmer and the user are two different people, 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. 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 in attracting programmers (Lattemann and Stiglitz 2006, Krishnamurthy and Tripathi 2006, Vujovic and Ulhøi 2006). As in the case of cost-saving characteristics, there are cost-raising characteristics that increase the TCO of OSS. In particular, the undirected innovation process leads to higher total costs of use of OSS. The 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.

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. Third, we include the TCO of 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 focussed solely on the 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, 2000b; Torvalds and Diamond, 2001; Lerner and Tirole, 4 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 the technology became the crucial determinant (see Bresnahan and Greenstein (1999). 12 2002, Bitzer et al., 2004; Bitzer and Schr¨oder, 2005; Hertel et al. 2003, Lakhani and Wolf 2005, Hars and Ou 2002). Hence, even though a profit motive can be ruled out for OSS developers, they nevertheless maximize these other payoffs. 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 products. 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 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 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 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.5 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 5 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. 13 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 software includes the entire bundle of its technological characteristics (Bessen, 2006), see Bitzer and Schr¨oder (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, e.g. 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, and switching costs (CSC 2004). In particular, the latter two are important characteristics of the software market (Schmidt and Schnitzer 2003). 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 6 It is always possible to overcome network externalities by corresponding investments, e.g., 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. 14 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 that specific point in time.7 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, i.e. innovation.

4. WAYS OPEN SOURCE GIVES YOU A COMPETITIVE EDGE

Building a tech stack is a major decision for every organization. While picking the right tools will set your team up for success, picking the wrong solutions or platforms can have devastating effects on productivity and profitability. To succeed in today's fast-paced world, organizations must make smart investments in digital solutions that enable them to move faster and increase operational agility. This is precisely why more and more organizations of all sizes and across all industries are embracing open source solutions. According to a recent McKinsey report, open-source adoption is the biggest differentiator for top-performing organizations.

Here are four reasons why adopting open source technology can help organizations drive competitive advantage and experience better business outcomes.

4.1. Extensibility and flexibility

Suffice it to say the world of technology moves quickly. For example, Kubernetes didn't exist before 2014, but today, it's impressively ubiquitous. According to the CNCF's 2020 Cloud-Native Survey, 91% of teams are using Kubernetes in some form.

More Great Content

- Free online course: RHEL technical overview
- Learn Advanced Linux Commands
- Download Cheat Sheets
- Find an Open Source Alternative
- Read Top Linux Content
- Check out open-source resources

One of the main reasons organizations are investing in open source is because it enables them to operate with agility and rapidly integrate new technologies into their stack. That's compared to the more traditional approach, where teams would take quarters or even years to vet, implement, and adopt software, making it impossible for them to pivot with any sense of urgency.

Since open source solutions offer complete access to source code, teams can easily connect the software to the other tools they use every day.

Simply put, open source enables development teams to build the perfect tool for what is at hand instead of being forced to change how they work to fit into how inflexible proprietary tools are designed.

4.2. Security and high-trust collaboration

In the age of high-profile data breaches, organisations need highly secure tools that enable them to keep sensitive data secure.

When vulnerabilities exist in proprietary solutions, they're often undiscovered until it's too late. Unfortunately for the teams using these platforms, the lack of visibility into source code means they're essentially outsourcing security to the specific vendor and hoping for the best.

Another main driver of open source adoption is that open source tools enable organisations to take control of their own security. For example, open-source projects—particularly those with large communities—tend to receive more responsible vulnerability disclosures because everyone using the product can thoroughly inspect the source code.

Since the source code is freely available, such disclosures often come with detailed proposed solutions for fixing bugs. This enables dev teams to remedy issues faster, continuously strengthening the software.

In the age of remote work, it's more important than ever for distributed teams to collaborate while knowing that sensitive data stays protected. Since open source solutions allow organizations to audit security while maintaining complete control over their data, they can facilitate the high-trust collaboration needed to thrive in remote environments.

4.3. Freedom from vendor lock-in

According to a recent study, 68% of CIOs are concerned about vendor lock-in. They should be. When you're locked into a piece of technology, you're forced to live with someone else's conclusions instead of making your own.

Proprietary solutions often make it challenging to take data with you when an organization switch vendors. On the other hand, open-source tools offer the freedom and flexibility needed to avoid vendor lock-in and take data wherever an organisation wants to go.

4.4. Top talent and community

As more and more companies embrace remote work, the war for talent is becoming even more competitive.

In the world of software development, landing top talent starts with giving engineers access to modern tools that enable them to reach their full potential at work. Since developers increasingly prefer open source solutions to proprietary counterparts, organisations should strongly consider open source alternatives to their commercial solutions to attract the best developers on the market.

In addition to making it easier to hire and retain top talent, open-source platforms also enable companies to tap into a community of contributors for advice on how to walk through problems and get the most out of the platform. Plus, members of the community also contribute to open source projects directly.

5. HOW TO COMPETE AGAINST OPEN SOURCE COMPETITION

5.1. Time vs Money

At one point during college I had $6 in my checking account. This lasted for about three months.

I remember riding across campus, about a 15 minute ride (each way) to save $.50 on a slice of pizza. This is inconceivable in my life today. Back then, time was abundant and money was scarce.

Then I graduated and got a job. At a salaried job making $80k plus benefits, your time is worth around $55/hour. Suddenly that ride across campus to save $.50 doesn't seem like the smart money decision it once was.

And thus it is with the majority of open-source software: Open source software is free if your time is worth nothing.

I'm bracing my inbox for emails from disgruntled Gimp users explaining how charging for software is bad, commercial software is evil, and my mother dresses me funny. But I don't buy it.

I've used mainstream image editors like Photoshop, Paint.NET and Gimp; some of my best friends are mainstream image editors. And when I saw Gimp I almost went blind. Children were weeping; the fruit was bruising.

Are there exceptions in the open-source world? Absolutely.

When an open-source project gets enough talented people working on it, it can become a downright masterpiece.

Firefox rocks. WordPress is awesome. Paint.NET rules. And Linux is pretty cool, though the lack of drivers and ease of use as a mainstream desktop OS after all this time is still a disappointment.

And yes, I know about Ubuntu. I also know that every friend to whom I've recommended it has run into major compatibility issues or a complete lack of drivers. Ubuntu is free, after 6 hours of research and command line tricks trying to get your laptop to connect to your network.

Have you ever tried Gimp? Or the admin control panel in Zen-Cart? Or tried to install a Perl or PHP module that didn't come out of the box? I've been a web developer for 10 years and I cringe when I see that I need a module that's not included, there goes two hours of my day searching, configuring and installing dependencies.

As a developer, I've probably had contact with 300 open source projects, components, and applications. I estimate that 80% of them required substantially more time to install, use, or maintain than their commercial counterparts. But depending on the price and feature set of their commercial counterparts, sometimes it's worth using the open-source app and sometimes it's not. $299/user for Vault vs. $0/user for Subversion? It depends on how badly you need live support and guaranteed bug fixes.

5.2. The Differentiators

The areas that kill the most time when consuming open source software are:

- Installation process
- Documentation
- Support
- Usability

I'm sure we can all point out a handful of open source projects that have decent documentation and decent usability. The vast majority do not. Even fewer can be installed in five minutes or less, even by an experienced software developer.

5.3. How to Compete Against Open Source

As a commercial software vendor you have to focus on your key advantages over open-source software:

1. Save Your Users Time. Ensure a painless installation process, top-notch documentation, top-notch support, and a minimal learning curve for getting started using your application.
2. Market Hard. You have a marketing budget; odds are high that your open-source competitor does not. If you can position your product well and build a reputation for good documentation, support and usability, you will sell software.
3. Focus on Features for Your Demographic. Your open-source competitor is going to win when it

comes to college students, hobbyists, and other groups where time is worth a lot less than licensing cost. You will have an edge with business users since time is highly monetized for entrepreneurs and enterprises. Build features for people who are likely to buy your product.

6. DISCUSSION

6.1. The Worst-Case Scenario for PS and OSS-SS Vendors

The worst-case scenario for both the PS and OSS-SS vendors is when OSS is as usable as PS and OSS. In such a scenario there is no incentive for the PS and the OSS-SS vendors to enter the software market. The OSS-SS vendors stay away because they face zero demand (even though they are highly usable) when they compete against freely available and highly usable OSS. On the other hand, both PS and OSS-SS will have a positive demand. However, the equilibrium price for PS is not enough to cover the marginal cost of selling the software, resulting in negative profits for the PS vendor (See Appendix B). Therefore, both PS and OSS-SS vendors have little incentive to improve the usability of freely available OSS. However, such a scenario begs an important question, i.e. what is the likelihood of this scenario (i.e. the usability of OSS application improves to the extent that it is comparable to that of PS applications) ever happening? For this to happen, the developers involved with the OSS project will have to give as much importance to the usability of the application as to its utility or functionality, and will have to develop user-friendly 0.1 0.2 0.3 0.4 0.5 0.6 0.7 Unusability 0.02 0.04 0.06 0.08 OSS SS Profits (0) * π S θ = Figure 3b: Equilibrium Profits in Software Markets with Low Network Benefits [Assume:V =1.5;c = 0.2] (β) * β S 20 interfaces and features for the OSS application. Though theoretically possible, a realistic assessment of OSS projects shows that this is not likely to happen soon. OSS application developers have traditionally favored function over form, which is evident in the lack of user-friendly features (e.g. user interfaces) in these applications and there is no evidence that this is changing (Nichols and Twidale 2003). One reason is that in most open source projects, the developers of OSS applications are also its core users (Mockus et al. 2000, 2002). For these developer-users, the OSS application is already " user-friendly" because they are highly skilled software professionals (Koch 2003), and they already know the software inside out (the source code is available to them) to modify it as they choose fit. In addition, most OSS developers work under a licence that prevents them from selling the code, so they develop features that are important to them and not necessarily to the market. Finally, there is hardly any mechanism to capture the feedback of average users who are not involved with the OSS application development process but are most in need of user-friendly features. Therefore, we can safely assume that the likelihood of the worst-case scenario ever happening is low.

6.2. How can commercial proprietary software producers compete against open source software?

This is one of the key questions that we started the paper with. In this paper, we divide the software markets into two broad categories- those that are characterised by strong network effects and those that are characterised by weak network effects. A vendor of PS software needs to identify the software market that it is competing in and act accordingly. If it is competing in software markets with strong network benefits, then it will remain an important player in the software market as long as OSS remains highly unusable and 21 OSS-SS remains unusable relative to the PS (see Figures 1, 2, and 3a). This makes the software markets such as e.g. the desktop office productivity software market, and desktop operating systems, attractive for PS vendors. Furthermore, a PS vendor can improve its competitive position by ensuring relatively high usability in its own products. In addition, they need to ensure very low usability in freely available OSS. They can do so by participating in OSS projects and encouraging the developers of OSS to focus their development efforts only on the functionality and reliability of open-source software. If the OSS becomes as usable as OSS-SS, then the OSS-SS vendor is forced to improve its own usability (see Figure 3a) or exit the market (see Appendix C). If the OSS-SS vendor decides to improve its usability further, then all software ends up with the same usability resulting in negative profits for PS, and no demand for OSS-SS (See Appendix B). If the PS vendor is competing in software markets characterized by weak network effects, then it should work towards improving the usability of OSS so that it is as usable as OSSS. In such a market OSS-SS has two options - (a) to improve its usability in relation to OSS, and (b) to exit the market (see Appendix C). If OSS-SS decides to improve its usability, it is constrained by the relationship

between its usability and profitability (see Figure 3b), which ensures that OSS-SS is never as usable as PS (since OSS-SS profits are maximised at 0 1 * < β = β S <). If possible, the PS vendor should help OSS developers to improve their usability to the level of OSS-SS (i.e.) * β S . This would force the OSS-SS vendor to exit the software market (see Appendix C). However, if the usability of OSS-SS is very low, then the PS vendor is better off by ensuring that OSS has the lowest usability among the three (see Figure 4) so that OSS-SS remains in the software market. This is an interesting result since it identifies the condition under which the presence of an OSS-SS vendor actually helps the competitive position of the PS vendor.

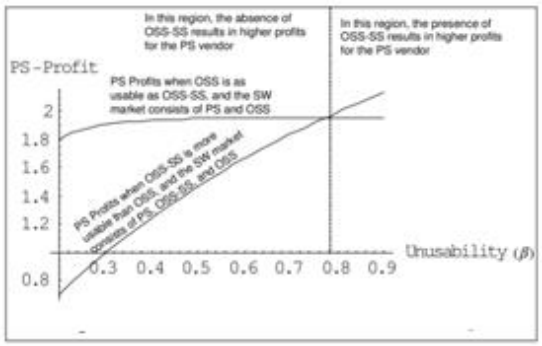

Fig 1: PS Profit in software markets with low NW benefits [1]

6.3. How can OSS-SS vendors compete against well established proprietary software?

OSS-SS vendors who sell usable OSS plus support services face a very competitive environment. On one hand, they have to compete with highly usable PS options, and on the other hand, they have to compete with freely available OSS. To compete successfully, the OSS-SS vendor needs to benchmark its usability against the competing PS. If it is competing in the software market characterized by weak network effects, then it is better off by ensuring that OSS-SS is highly usable compared to the OSS, but not as usable as the competing PS (see Figures 1, and 3b). On the other hand, if it is competing in software markets characterized by high network effects then it needs to offer a product that is at least as usable as the competing PS. By doing so it can force the PS vendor out of the software market (see Figure 1 and Figure 3a). OSS-SS vendors also need to keep in mind that users differentiate between OSS and OSS-SS mainly on the basis of usability. If OSS starts to become more usable, then OSS-SS vendors lose this strategic advantage. Therefore, OSS-SS need to ensure that the " free" open-source software is not very useful when compared to the usability of PS or OSS-SS. This can be achieved by actively participating in relevant open source projects and having enough influence among the developers to focus most of their development efforts on improving software functionality and reliability. In addition, they should provide a highly usable version of OSS only if the users also buy the support services. For instance, some OSS-SS vendors continue to provide highly usable versions of relevant OSS for "free" (e.g. Suse 9 by Novell is a highly usable Linux flavour for desktops and it is available for free download). This strategy will have an adverse impact on their market shares and profitability in relevant software markets.

7. 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 the 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 analyze 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 analyze how different cost structures influence the decision of the agents to innovate or not. Thus, we reexamine the often 23 assumed cost advantages 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 lead 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.

REFERENCE

[1] Ravi Sen, A Strategic Analysis of Competition Between Open Source and Proprietary Software https://econwpa.ub.uni-muenchen.de/ econ-wp/io/removed/0510004.pdf

[2] Philipp J.H. Schröder, Open Source Software, Competition and Innovation.

[3] Rob Walling, https://robwalling.com/2009/ 08/11/how-to-compete-against-open-source-competition/

[4] Jason Blais, https://opensource.com/article /21/4/open-source-competitive-advantage

[5] Bessen, James (2006): Open Source Software: Free Provision of Complex Public Goods, in J. Bitzer and P.J.H. Schr¨oder (eds.): The Economics of Open Source Software Development, Elsevier, pp. 57-82.

[6] Bitzer, J¨urgen (2004): Commercial versus open-source software: The role of product heterogeneity in competition, Economic Systems, Vol. 28, No. 4, 2004, pp. 369-381.

[7] Bitzer, J¨urgen, Wolfram Schrettl and Philipp J.H. Schr¨oder (2004): Intrinsic Motivation in Open-Source Software Development, Discussion Paper, No. 2004/19, Department of Economics, Free University Berlin.

[8] Bitzer, J¨urgen and Philipp J.H. Schr¨oder (2005): Bug-Fixing and CodeWriting: The Private Provision of Open-Source Software, Information Economics and Policy, Vol. 17, No. 3, 2005, pp. 389-406.

[9] Bitzer, J¨urgen and Philipp J.H. Schr¨oder (2006): The Impact of Entry and Competition by Open-Source Software on Innovation Activity. In: The Economics of Open-Source Software Development, J. Bitzer and P.J.H. Schr¨oder (eds.), Elsevier Science Publishers, pp. 219-246.

[10] 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.

[11] Casadesus-Masanell, Ramon and Pankaj Ghemawat (2006): Dynamic Mixed Duopoly: A Model Motivated by Linux vs. Windows, Management Science, Vol. 52, No. 7, pp. 1072-1084.

[12] Computer Sciences Corporation (CSC) (2004): Open Source: Open for Business, [http://www.csc.com/features/2004/uploads/L EF OPENSOURCE.pdf] , date 25. July 2006. 25

[13] Dahlander, L. and M. G. Magnusson (2005): Relationships between open-source software companies and communities: Observations from Nordic firms, Research Policy, Vol. 34, No. 4, pp. 481493.

[14] Dahlander, Linus and Mats G. Magnusson (2006): Business models and community relationships of open source software firms, in J. Bitzer and P.J.H. Schr¨oder (eds.): The Economics of Open Source Software Development, Elsevier, pp. 111-130.

[15] Economides, Nicholas and Evangelos Katsamakas (2006a): Linux vs. Windows: A comparison of application and platform innovation incentives for open source and proprietary software platforms, in J. Bitzer and P.J.H. Schr¨oder (eds.): The Economics of Open Source Software Development, Elsevier, pp. 207-218.

[16] Economides, Nicholas and Evangelos Katsamakas (2006b): Two-sided competition of proprietary vs. open source technology platforms and the implications for the software

industry, Management Science, Vol. 52, No. 7, pp. 1057-1071.
[17] 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.
[18] 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.
[19] Haefliger, Stefan, Georg von Krogh, and Sebastian Spaeth (2006): Knowledge Reuse in Open Source Software, Working paper ETH Zurich. Hars
[20] A. and S. Ou (2002): Working for Free? Motivations for Participating in Open-Source Projects, International Journal of Electronic Com26 merce, Vol. 6 (3), pp. 25-39.
[21] 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.
[22] 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.
[23] Kogut, Bruce and Anca Metiu (2001): Open-Source Software Development and Distributed Innovation, in: Oxford Review of Economic Policy, Vol. 17, No. 2, pp. 248-264.
[24] 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]. Krishnamurthy, Sandeep and Arvind K.Tripathi (2006): Bounty Programs in Free/Libre/Open Source Software (FLOSS), in J. Bitzer and P.J.H. Schr¨oder (eds.):
[25] The Economics of Open Source Software Development, Elsevier, pp. 165-184. Kuan, J., (2001): Open Source Software as Consumer Integration into Production, Working Paper, [http://ssrn.com/abstract=259648], date 29 July 2004.