Dual Licensing in Open Source Software Markets*

Stefano Comino† Fabio M. Manenti‡

May 2007

Abstract

Dual licensing has proved to be a sustainable business model for various commercial software vendors employing open source strategies. In this paper we study the main characteristics of dual licensing and under which conditions it represents a profitable commercial strategy. We show that dual licensing is a form of versioning, whereby the software vendor uses the open source licensing terms in order to induce commercial customers to select the proprietary version of the software. Furthermore, we show that the software vendor prefers dual licensing to a fully proprietary strategy when the customers are very sensitive to the reciprocal terms of the open source license.

J.E.L. codes: L11, L17, L86, D45.

Keywords: open source software, dual licensing, copyright, versioning, forking.

*Much of this work has been conducted while the authors were visiting the School of Information of the University of California at Berkeley. We are extremely grateful to Hal Varian for his hospitality. Financial support from "Progetto di Ateneo" - Padova, 2005.

†Dipartimento di Economia, Università di Trento, Via Inama 5, 38100 TRENTO (Italy), Tel. (39) 0461 882221, email: stefano.comino@economia.unitn.it

‡Corresponding author: Dipartimento di Scienze Economiche "M. Fanno", Università di Padova, Via del Santo 33, 35123 PADOVA (Italy), Tel. (39) 049 8274238, Fax. (39) 049 8274211, email: fabio.manenti@unipd.it

1 Introduction

Until recently, open source (OS) was seen unfamiliar by the business community and, in many cases, it was perceived as a real threat to commercial vendors. In the very last years, things have changed substantially and both large established incumbents such as IBM, HP or NEC as well as start-ups are increasingly embracing OS strategies (Hecker, 1999; Bonaccorsi et al., 2006; Rajala et al., 2007).

Commercial firms may enjoy several benefits by "going open source". For instance, a firm may take advantage of the feedbacks received from independent developers in terms both of bug fixing and code enhancement. Furthermore, open source represents a powerful channel of software distribution: it may constitute a key strategic instrument to improve the perceived quality of the product and to enlarge the installed base of users, thus helping establishing an industry standard. Also, the decision to release the source code to the OS community might be necessary in order to cope with an increased competition in the marketplace.[1]

The key issue for a software vendor is how to design a sustainable business model based on open source solutions, provided that various features of OS software development and distribution seem to be unappropriate for commercial exploitation.[2] In other words, it is important to understand if and how it is possible to adhere to the principles and the rules inspiring the OS movement while still creating enough cash to make OS viable for commercial vendors. A typical commercial strategy that has been successfully employed is to sell complementary products and to profit from OS-related segments[3] and/or services.[4]

[1] According to many commentators, Netscape started the Mozilla project due to its inability to meet the competitive threat posed by Internet Explorer; see West and Gallagher (2004).

[2] For instance, OS licenses require the code of the software to be freely re-distributable; when releasing the software code, an individual, or the firm cannot prevent or restrict (e.g. by requiring royalties) its re-distribution. More specifically, article 1 of the Open Source definition states that "The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale." (See the OSI definition WEB site: `www.opensource.org/docs/definition.php`).

[3] In 2001, IBM started the open source project Eclipse in order to promote the use of Java development tools for server products. The company profited from selling related products such as components of WebSphere and WebLogic; see West and Gallagher (2004).

[4] This is often the case of several software vendors that offer deployment support, customization and adds-on products for Linux solutions. Rajala et al. (2007) provide a recent and comprehensive discussion of the Red Hat case.

In this paper we focus on an alternative business model that is gaining popularity within software vendors that adopt OS solutions which is known as *dual licensing*.[5] With dual licensing firms mix traditional and OS-based strategies by offering one core product under two different licensing schemes: the software is offered to users both under a traditional proprietary license as well as under an open source one. Michael Olsen, CEO of Sleepycat Software Inc., producer of the embedded database BerkeleyDB, describes the dual licensing strategy of his company as follows:

> *"The Sleepycat open source license permits you to use Berkeley DB [...] at no charge under the condition that if you use the software in an application you redistribute, the complete source code for your application must be available and freely redistributable under reasonable conditions. If you do not want to release the source code for your application, you may purchase a license from Sleepycat Software."* (Välimäki, 2005 pp. 209-210)

According to Välimäki (2005), this scheme represents the prototypical example of dual licensing strategy. The OS version of the software is available free of charge, and it is distributed under very restrictive licensing terms.[6] The customer who downloads the OS version of the code has to adhere to the strong reciprocal provisions on derived works imposed by the OS license: if she/he modifies the downloaded code to create new software, the new software code has to be made freely available and re-distributable. Alternatively, the customer can obtain the code under a proprietary license; in this case the customer is required to pay a fee but is freed from the reciprocal provision, being no longer obliged to publish

[5] Välimäki (2005) presents in details three well known examples of companies that have succesfully adopted a dual licensing strategy: Sleepycat, MySQL and Trolltech. On the Sleepycat experience see also Olson (2006). According to a recent paper by Koski (2007), nearly 6% out of a sample of 270 OS software business firms in Finland, Germany, Italy, Portugal and Spain, employ dual licensing strategies.

[6] The licensing terms set the rules under which the code is developed and distributed and represent the core of the OS model. Currently, there are more than 40 different licensing schemes which differ mainly with respect to the restrictions (also known as the degree of reciprocity) that they impose on derived works. At the two extremes there are the GNU General Public License (GPL) and the Berkeley Software Distribution (BSD) license. GPL-like licenses are strongly reciprocal since they impose that any derived work has to be distributed under the same licensing terms as the original software (copyleft or inheritance provision). On the contrary, under BSD-like licenses, derivative works need not be licensed under the same terms. In between these two extremes, there exist several OS licenses that mandate different degrees of reciprocity. On these points see Bonaccorsi and Rossi (2003), Rosen (2001), and Lerner and Tirole (2005).

any modification of the source code.

As a matter of fact, dual licensing seems an appropriate strategy when a relevant part of the demand is generated by commercial users that need the software in order to embed it into their own products. These "embedders" employ the source code of the software as an input in order to produce other software; the derived software may be a product per se or, more frequently, it may constitute part of the technology of a more complex system produced and sold by the embedder. In both cases, it is clear that since an embedder wants to keep proprietary control on the derived product, it prefers to receive (i.e. it is willing to pay for) the source code he/she needs to embed under a proprietary licensing scheme that does not impose reciprocity.[7]

The choice of releasing the source code to the OS community implies some risks too: although the copyright is still in the control of the software vendor that has originally developed the code,[8] the act of placing the software on a public repository may stimulate potential competition. Competition may come in different forms; on the one hand, being the OS version of the software freely downloadable from the internet, there is always the risk that the free version *cannibalizes* the market. On the other hand, the original project may be taken over by external programmers that start independent development on it, thus creating a distinct and competitive piece of software that possibly overcomes in terms of quality or adoption the original one.[9] In the industry jargon, this threat is known as *forking*.

In order to make dual licensing a viable strategy, the software vendor must take actions to protect her business from these threats. On the one hand, adds-on and additional features and functionalities bundled into the proprietary version of the code represent a possible

[7]This explains the fears of the many companies that, for example, embed OS Linux into their own platforms but that decide not to release their derived code under the GPL, because it represents the bulk of their product's core technology. They fear that some day a court may oblige them to adhere to the GPL reciprocal provision and to release their derived software to the public domain. This fear has been exacerbated since the well known "SCO vs IBM" case, whereby IBM has been alleged to violating SCO's intellectual property rights by distributing a Linux distribution with copied code; see Moglen (2003).

[8]It is important to stress that by releasing the OS version of the software, a vendor does not give up the copyright on the code she has written; more precisely, at any moment in time, the original developer can use the lines on which she has the copyright in order to offer a proprietary version of the software.

[9]This is what has happened, for instance, to SSH Communications Security Corp and to its SSH secure shell protocol; see Välimäki (2005).

safeguard against cannibalization;[10] on the other hand, in order to reduce the risk of forking, the software vendor should try to keep control of the OS version of the project e.g. by becoming the project manager of the version placed in the OS repository.[11]

In this paper we consider a profit maximizing software vendor that starts developing a certain software code; the code is useful to commercial customers (the "embedders") that embed the source code into their own products.

The software vendor has to decide whether to realize a fully proprietary version of her product or to release the code to the OS community and to employ a dual licensing strategy. In this latter case, the software firm obtains the collaboration of independent developers of the OS community; furthermore, she also benefits from a larger installed base of users. However, by going open source, the vendor gives away her monopolistic position: embedders may obtain the code directly from the OS repository, or they can buy it from independent developers that, obtained the original code, may have started redistributing a modified version of the software.

In the paper we show that when the code is released to the OS community, the software vendor will distribute it under a strongly reciprocal OS license; this is due to the fact that commercial customers, i.e. those that want to embed the software code into their products, are interested not only in the quality of the code that they obtain, but also in its licensing terms: the more reciprocal the licence, the more difficult for them to extract profits from their derived products (i.e. they face the risk of being accused of violating OS licensing). By releasing the OS code under strongly reciprocal licensing terms, the software vendor makes the OS version less attractive to commercial customers, thus reducing the competitive threat. In this sense, dual licensing is a form of versioning, where the software vendor uses the license in order to induce the embedders to prefer the proprietary version of the code even though

[10]Note that this behavior is not always effective since it might go against the "ethical" principles of the OS community which might feel exploited being released a downgraded version of the software code (Capobianco, 2006).

[11]Similarly, the software vendor should carefully manage her copyright on the software e.g. by acquiring the copyright on the lines of code added by independent developers, or by checking and rewriting them before inclusion into the proprietary version. According to Välimäki (2005), in order to avoid the dilution of copyright ownership, in MySQL AB "All contributions are checked and rewritten by company developers ..." (p. 212).

this is sold at a positive price.[12]

We show that dual licensing is optimal only when commercial customers are very sensitive to the reciprocal provisions of the OS license. A customer may be more (resp. less) sensitive to the degree of reciprocity of the OS licence either because he does (does not) need to embed the software into his product or because the original software is (is not) thought to be embedded into further technology. For example, by its very nature a packaged/mass market software is not made to be embedded: in this case the licensing terms under which it is distributed do not matter and dual licensing cannot be a viable strategy provided that the OS version of the software would inevitably cannibalize the proprietary one. On the very opposite, when the derived software is the core of the embedders technology, the reciprocal provisions imposed by the OS license do very much disturb embedders who will pay to obtain a proprietary copy of the code; in this case the software vendor will find it optimal to employ a dual licensing strategy.

The rest of the paper is structured as follows: in Section 2 we present the outline of the model and the main assumptions, while in Section 3 we derive the main results of our analysis. Section 4 concludes.

2 The model

Suppose that a commercial software house (the original developer, OD) has started developing a new software; the product is directed mainly to other commercial firms that need the source code of the software to embed or to include it into their own products. We call these latter as the "embedders", and we assume that they have mass 1. Due to the their interests in using the code into their own commercial products, the embedders are interested not only in the quality of the code "per-se", but also to the terms of licensing under which the source code is distributed: since commercial customers want to keep proprietary control on their products, they prefer to receive the code under unrestrictive licenses, namely those licenses that do not impose strong reciprocal provisions.

Let us assume that the completion of the software proceeds through two successive stages: a development phase and a distribution phase. In the first stage, the source code is written

[12]For details on the description and implementation of versioning strategies, see Shapiro and Varian (1998) and Bhargava and Choudhary (2001).

and bugs are fixed; in the following stage, the OD bundles the code with additional services (manuals, customer care services, warranties etc.) and/or she customizes the product to specific users needs. Once the code is completed, the OD fixes the price of its product.

The software house can complete the project on her own, realizing a fully proprietary version of the product; alternatively, at the beginning of each stage, she can opt to release the code to the open source (OS, hereafter) community in order to obtain feedbacks from independent developers and to improve the quality of her product. During the development phase, the community cooperates directly on writing and improving the code of the software, while in the distribution stage its main effect is to enlarge the installed base of users of the software, thus generating positive network effects. We call the first type of effect as *OS development externality* and the latter type as *OS network externality*. We assume that the OS community contributes to software development and distribution but it is not willing to pay for its use/adoption.

However, placing the code on an OS repository has another notable consequence: any individual may freely download the software code. This implies that not only embedders can obtain an OS version of the software at no cost, but also that other independent developers may download, modify and sell the code in the attempt to exploit profit opportunities, i.e. the so called "forking" may take place.[13] We call these latter independent developers as "redistributors". The market of redistributors is "de-facto" perfectly contestable: given that the code is freely available to everyone, entry will occur whenever profitable. We assume that the original developer and the redistributors compete in prices.

Whenever the code is released as open source, the OD must also decide the software licensing terms. As clarified below, the decision on the type of OS licence is crucial since it affects both the market value of the software and the extent to which the OD can appropriate the contributions provided by the independent developers: the more restrictive the licence, the less the original developer benefits from the contribution of the OS community.

Figure 1 provides a graphical description of the choices of the original developer in the process of software completion:

- in the development stage, the original developer chooses whether to release the code to

[13]More precisely, in software engineering, forking occurs when developers obtain the source code from one software package and start independent development on it, creating a distinct piece of software.

the OS community or not; in this latter case, the firm needs also to decide the licensing terms of the OS project, formally she chooses the degree of reciprocity $x \in [0, 1]$. $x = 0$ is the case of a fully non reciprocal OS licence while, on the extreme opposite, $x = 1$ represents the highest level of reciprocity.

- In the distribution stage:

  a) if the code has not already been released to the OS community, the OD may choose to release it at this stage. If, again, she chooses not to release the code, the software is kept proprietary; alternatively, if the code is released to the OS community, the OD needs to decide a licensing strategy: either to leave the code OS only or to employ a dual licensing scheme by offering also the proprietary version.

  b) if the code has been released at the development stage, then the OD needs only to decide wether to leave the code OS only or to employ a dual licensing scheme.

- Finally, any time the OD offers a proprietary version (either alone or as a part of a dual license scheme), she sets a price for the code.

[Figure 1 about here]

2.1 Quality of the software and agents' preferences

Let us denote with q_P the quality of the proprietary version of the software offered by the original developer; software quality depends on how the software has been developed and distributed. There are three possible patterns: *i*) the software code is never released to the OS community, *ii*) it is released at the development stage or, finally, *iii*) the OD releases the code only during the distribution stage.

Table 1 shows the quality of the software according to the chosen pattern:

	the code is never released	the code is released at the development stage	the code is released at the distribution stage
q_P	$v_1 + v_2$	$v_1 + (1 - x)\theta_1 + v_2 + \theta_2$	$v_1 + v_2 + \theta_2$

Table 1: OD software quality

where v_1 represents the contribution to the quality of the code by the original developer at the development stage, while v_2 is the additional value created by the OD in the distribution stage (i.e. customization/support services, etc.). Similarly, θ_1 is the contribution of the OS community to the quality of the code (development externality); when the OD releases the software to the OS community at the development stage, the amount of the development externality effectively enjoyed by the original developer depends on the degree of reciprocity of the OS licence x: the less reciprocal the licence, the easier to incorporate the contributions from the OS community into the OD software code. Finally, θ_2 represents the network effect generated by the adoption of the software by the OS community.[14]

Whenever the OD donates the code to the open source community, any individual can freely obtain the code and redistribute it. Therefore, in this case the embedders have two additional ways of obtaining the code: *i*) download it directly from the OS repository (we refer to this as the "downloadable" version) or *ii*) buy it from the competitive redistributors; these latter get the code from the repository and offer their own (still OS) version of the software after having added some customization features (we call this the "redistributed" version).

The following table describes the quality of the alternative versions of the software available to the embedders: q_D is the quality of the code at the repository and q_R is the quality of the redistributed version.

	the code is never released	the code is released at the development stage	the code is released at the distribution stage
q_D	n.a.	$v_1 + \theta_1 + \theta_2$	$v_1 + \theta_2$
q_R	n.a.	$v_1 + \theta_1 + \beta v_2 + \theta_2$	$v_1 + \beta v_2 + \theta_2$

Table 2: software quality of the alternative versions

Consider q_D first; when the code is released at the development stage, the quality of the OS version fully incorporates both the development and the network effects, θ_1 and θ_2. Clearly, having the OD released her code at the development stage, the quality of the OS version in this case incorporates v_1 but not the OD customization/adds-on services v_2. When

[14] As a typical example of usage externality, as long as standardization is ensured, it seems natural to assume that the benefit enjoyed by the OD when the network of users gets larger does not depend on the type of OS license.

the code is released only at the last stage, q_D does not include θ_1 given that the community has not been given the opportunity to contribute to its development.[15]

Finally, q_R is simply q_D plus the additional quality provided by redistributors through their customization/adds-on services. We denote with βv_2 the quality added by the redistributors, with $\beta \geq 0$; the larger β, the more capable redistributors in customizing and packaging the software.[16]

The embedders' gross utility from adopting a software depends both on the quality of the code and on the degree of reciprocity imposed by the licensing terms. The gross utility from downloading the source code from the OS repository is given by:

$$U_D(q_D, x) = q_D - tx,$$

where tx denotes the disutility incurred when the reciprocity of the OS license is x; t measures how much the degree of license reciprocity matters to the embedders and it depends on how these latter use the software, and, also, on its nature. Embedders use the OD software code as a production input to realize other, derived, software that they either sell directly or that they incorporate into their own products; t will be larger the more the derived software represents the core of the embedders' products/technologies: the more relevant the derived software code in the embedded system, the more penalized the embedder that might be forced by a court to release it under reciprocal licensing terms. Obviously, the relevance of software code within the embedders' technology depends also on the nature of the software commercialized by the original developer: by definition, embedded software is characterized by a large t, while mass market software applications (e.g. text editors) have a small, possibly zero, value of the parameter t.

When adopting the software offered by a redistributor, embedders enjoy a gross utility

[15]It should be noted that we are implicitly assuming that the development externality cannot be incorporated into the OS software code in later stages; this amounts to say that once the development stage has been completed, the code cannot be developed any further.

[16]We are implicitly assuming that the proprietary and the OS versions of the software are compatible, thus benefitting of the same amount of network externalities. Clearly, forking may conduct to the creation of incompatible OS versions; in this case the network externalities of the proprietary and OS versions of the software would be different. The analysis of this case is qualitatively similar to the one we propose, with the parameter β mimicking the role of the externalities: a larger installed base for the OS versions could be represented by a β larger than 1.

$U_R(q_R, x) = q_R - tx$: they take advantage of the quality q_R but, again, they are affected by the reciprocal provisions of the OS license chosen by the original developer.

Alternatively, embedders can buy the software from the OD obtaining the code under a proprietary license; when setting the terms of the proprietary license, the interests of the OD and of an embedder are convergent: they are both better off by selecting the licensing scheme that fulfills completely the needs of the embedder, i.e. a licence scheme that allows the embedder to keep full control on its derived product. This implies that the willingness to pay for a proprietary version is not affected by the disutility tx; formally, it is simply given by $U_P(q_P) = q_P$.

3 Market equilibrium

We solve the game by backward induction. Consider first the case in which the OD does never release the code to the OS community: embedders are offered only the proprietary version and their net utility from adopting it is $v_1 + v_2 - p_P$. Clearly, being the monopolist, the OD charges the embedders' reservation value, $p_P = v_1 + v_2$. Note that, since we are assuming that the mass of embedders is equal to 1, this price corresponds to the OD's revenues/profits.

In the following sections we analyze the remaining sub-games: *i*) the code is released at development and, *ii*) the code is released at distribution.

3.1 Code released at the development stage

If the OD has already released the source code to the open source community in the first stage, she is certainly better-off by offering the code also under a proprietary version, that is by employing a dual licensing strategy. We need to define the optimal price charged by the OD and her licensing choice x.

According to Tables 1 and 2, the net utility that in this case an embedder enjoys from buying at p_P the proprietary version offered by the OD is $v_1 + (1 - x)\theta_1 + v_2 + \theta_2 - p_P$. Alternatively, the utility from adopting the downloadable version is $v_1 + \theta_1 + \theta_2 - tx$, while the net benefit from buying at p_R the redistributed version is $v_1 + \theta_1 + \beta v_2 + \theta_2 - tx - p_R$.

Competition between redistributors drives p_R down to zero and it makes the redistributed

version always preferred to the downloadable one. Straightforward calculations show that the optimal price for the proprietary version is as follows:[17]

$$p_P(x) = \begin{cases} \text{any } p & \text{if } 0 \leq t < \theta_1 - \frac{(1-\beta)v_2}{x} \\ (1-\beta)v_2 + x(t-\theta_1) & \text{if } \theta_1 - \frac{(1-\beta)v_2}{x} \leq t < \frac{v_1+\theta_1+\beta v_2+\theta_2}{x} \\ v_1 + (1-x)\theta_1 + v_2 + \theta_2 & \text{otherwise} \end{cases} \quad (1)$$

When t is very small, the redistributed version is superior to the proprietary one; in this case the OD cannot sell its product whatever the charged price, and her profits are zero. The opposite situation occurs when t is so large that embedders never choose neither the downloadable nor the redistributed version: formally when $U_D(\cdot) < 0$ and $U_R(\cdot) < 0$. This amounts to say that the OD acts as a monopoly and charges the embedders' reservation price. For intermediate values of t, competition between the redistributed and the proprietary versions induces the OD to charge a price equal to the quality difference between the two.

We are now in the position to define the optimal degree of the OS license reciprocity when the OD releases the code at the development stage.[18]

Proposition 1. *When the code is released during the development stage, the optimal OS license is: i)* $x = 0$*, if* $t < \theta_1$*, ii)* $x = 1$*, if* $\theta_1 \leq t < v_1 + \theta_1 + \beta v_2 + \theta_2$*, and iii)* $x = \frac{v_1+\theta_1+\beta v_2+\theta_2}{t}$*, otherwise.*

The choice of the terms of licensing has two effects: on the one hand the more reciprocal the license, the less the OD benefits from the development externality since the contributions of the OS community are harder to incorporate into the proprietary version; this "development effect" is proportional to θ_1. On the other hand, a larger x makes the downloadable and the redistributed versions less competitive since they become less attractive for the embedders; formally, this "competitive effect" is proportional to the parameter t.

When $t < \theta_1$, the development effect dominates and the OD optimally sets $x = 0$ (case i)). On the contrary, for $t \geq \theta_1$, the competitive effect dominates and the OD strategically chooses highly reciprocal licensing terms to deteriorate the attractiveness of the alternative versions of her code. More precisely, when $\theta_1 \leq t < v_1 + \theta_1 + \beta v_2 + \theta_2$, the OD chooses a

[17] For the sake of simplicity, expression (1) shows the optimal price schedule when $\theta_1 - \frac{(1-\beta)v_2}{x}$ and $\frac{v_1+\theta_1+\beta v_2+\theta_2}{x}$ are positive and finite. In the formal analysis of Proposition 1, we fully take into account of all the possible cases.

[18] The proof of all the results are in the appendix.

fully reciprocal license, $x = 1$. For even larger levels of t, both the downloadable and the redistributed versions are very poor alternatives for the embedders; in this case, the OD finds it optimal to release the code under a non fully reciprocal license, $x < 1$: the OD selects the minimum level of reciprocity such that the alternative versions are not attractive for the embedders. The OD does not set x at a higher level since this would reduce even further the amount of development externalities that she would benefit.

3.2 Code released at the distribution stage

As in Section 3.1, when the OD releases the code at the distribution stage, she is certainly better-off under dual licensing. By using the same procedure employed in the previous section, the optimal price charged by the original developer is the following:[19]

$$p_P(x) = \begin{cases} \text{any } p & \text{if } 0 \leq t < \frac{v_2(\beta-1)}{x} \\ (1-\beta)v_2 + tx & \text{if } \frac{v_2(\beta-1)}{x} \leq t < \frac{v_1+\beta v_2+\theta_2}{x} \\ v_1 + v_2 + \theta_2 & \text{otherwise} \end{cases} \tag{2}$$

The next proposition describes the OD optimal licensing policy when the code is released at the distribution phase.[20]

Proposition 2. *When the OD releases her code at the distribution stage, it is always optimal to set* $x = 1$.

Proposition 2 is of immediate interpretation: when the code is not released at development, the OD does not benefit from the development externality and only the competitive effect matters; therefore, in this case the original developer finds it optimal to choose a fully reciprocal license.

3.3 The equilibrium of the game

It is now possible to describe the equilibrium of the game. For the sake of simplicity we focus on the case $v_1 \geq \theta_1$; the alternative scenario is qualitatively similar and it is omitted

[19] For the sake of simplicity, expression (2) shows the optimal price schedule when $\frac{v_2(\beta-1)}{x}$ and $\frac{v_1+\beta v_2+\theta_2}{x}$ are positive and finite. In the formal analysis of Proposition 2, we fully take into account of all the possible cases.

[20] The proof of Proposition 2 goes in the same way as the proof of Proposition 1 and it is omitted for brevity.

for brevity.

Proposition 3. *When $v_1 \geq \theta_1$, the original developer's optimal choices are: sell the code under proprietary version, when $t \leq v_1 + \beta v_2$; release the code at the distribution stage at $x = 1$ and dual license, when $t \in (v_1 + \beta v_2, v_1 + \theta_1 + \beta v_2 + \theta_2)$; release the code at the development stage at $x = \frac{v_1+\theta_1+\beta v_2+\theta_2}{t}$ and dual license, otherwise.*

This Proposition is interesting and shows how the optimal license strategy chosen by the original developer is a complex mix of decisions about if and when to release the code of her software to the OS community and, eventually, at which degree of reciprocity. When the code is released, the OD obviously opts to dual license in order to make money on its product.

The explanation of the result is very intuitive. When t is low enough, the downloadable and the redistributed versions of the code represent for the embedders two viable alternatives to the proprietary version; this implies that by releasing the code to the OS community, the original developer allows strong competitors to enter the market, thus cannibalizing her business. In this case, it is clear that the best strategy for the OD is to never release the code and to keep full proprietary control on her product.

As t gets larger, the two competing versions become less and less attractive to the embedders; this induces the original developer to release the code to the OS community in order to benefit from the externalities. In this context, the OS license scheme is chosen strategically: in order to protect her market, the original developer licenses the OS version under strong reciprocal terms even thou this mitigates the development externality she benefits.

More specifically, when t is not too large, the OD releases the code under a fully reciprocal OS scheme. The key decision here is when to release: by releasing the code early at the development stage, the OD induces an improvement in the quality of the competing alternatives by θ_1, while appropriating only of a share of it, $\theta_1(1 - x)$, for her proprietary version; therefore the OD finds it optimal to release the code later at distribution.

Finally, when t is sufficiently large, the embedders find the downloadable and the redistributed versions poor alternatives to the proprietary one; in this case, the OD releases the code already at the development stage and takes advantage of both the development and the network externalities. Notably, the code is released under a less than fully reciprocal license: the OD selects the minimum level of reciprocity such that the alternative versions

are not attractive for the embedders, i.e. when t is very large, the original developer is able to foreclose the competitors by choosing an appropriate degree of reciprocity of the OS license. Clearly, she does not set x beyond the foreclosure level since this would reduce the amount of development externalities that she would benefit.

4 Conclusions

In this paper, we have analyzed the behavior of a commercial software vendor that starts a new software project. The firm can choose whether to realize a fully proprietary version of the software or, instead, to release the code to the OS community and to employ a dual licensing strategy. We show that dual licensing is a form of versioning that the software vendor might use in order to benefit from the contributions of the OS community still being able to profit from the sale of the proprietary version of the software. We show that the software vendor uses the licensing terms strategically in order to reduce the competitive threat represented by the OS version of the code; namely, she releases the OS code under strongly reciprocal terms in order to induce commercial customers to adopt the proprietary version which is sold at a positive price.

We show that the software vendor prefers a dual licensing strategy when her customers are sensitive to the reciprocal provisions of OS licenses; to the contrary, she finds it more beneficial to realize a fully proprietary version of the code. In other words, dual licensing seems to represent a viable business strategy for embedded software and/or when the software code represents a central piece of technology for the platforms or systems that commercial customers produce and sell.

Appendix

Proof of Proposition 1. Note that $v_1+(1-x)\theta_1+v_2+\theta_2$ decreases with x, while $(1-\beta)v_2+x(t-\theta_1)$ decreases with x only if $t<\theta_1$. Therefore, when $t<\theta_1$, the OD sets $x=0$.

Consider now the case of $t\geq\theta_1$ and denote with $\tilde{x}=\frac{v_1+\theta_1+\beta v_2+\theta_2}{t}$, the value of x such that $(1-\beta)v_2+\tilde{x}(t-\theta_1)=v_1+(1-\tilde{x})\theta_1+v_2+\theta_2$; note that by choosing $x\leq\tilde{x}$ the OD obtains $(1-\beta)v_2+x(t-\theta_1)$, while by setting $x>\tilde{x}$, she gets $v_1+(1-x)\theta_1+v_2+\theta_2$. Therefore, the optimal licence is $x=min\{\tilde{x},1\}$; namely, the OD sets $x=1$, if $t<v_1+\theta_1+\beta v_2+\theta_2$, and $x=\frac{v_1+\theta_1+\beta v_2+\theta_2}{t}$ otherwise. □

Proof of Proposition 3. In what follows we consider the case of $\beta\leq 1$; the proof for the case of $\beta>1$ goes in the same way and it is omitted for brevity.

When the OD sells the code only under the proprietary version, she obtains v_1+v_2; whenever she releases the code to the OS community, she optimally follows a dual licensing scheme. From Subsection 3.1, we know that when the OD releases the code at the development stage, her profits are as follows:

- when $t\leq\theta_1$, the OD obtains $(1-\beta)v_2$;

- when $t\in(\theta_1, v_1+\theta_1+\beta v_2+\theta_2]$, profits are $(1-\beta)v_2+t-\theta_1$;

- when $t>v_1+\theta_1+\beta v_2+\theta_2$, profits are $v_1+(1-\tilde{x})\theta_1+v_2+\theta_2$, where $\tilde{x}=\frac{v_1+\theta_1+\beta v_2+\theta_2}{t}$.

From Subsection 3.2, when the OD releases the code at the distribution stage, her profits are as follows:

- when $t\leq v_1+\beta v_2+\theta_2$, profits are $(1-\beta)v_2+t$;

- when $t>v_1+\beta v_2+\theta_2$, OD's profits are $v_1+v_2+\theta_2$.

From a simple comparison of the profits in the three cases, Proposition 3 immediately follows. □

References

Bhargava, H. and Choudhary, V. (2001). Second Degree Price Discrimination for Information Goods under Nonlinear Utility Functions. In *HICSS '01: Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 7*, page 7039, Washington, DC, USA. IEEE Computer Society.

Bonaccorsi, A., Giannangeli, S., and Rossi, C. (2006). Entry Strategies under Competing Standards: Hybrid Business Models in the Open Source Software Industry. *Management Science*, 52(7):1085–98.

Bonaccorsi, A. and Rossi, C. (2003). Licensing Schemes in the Production and Distribution of Open Source Software. An Empirical Investigation. Sant'Anna School of Advanced Studies, Pisa.

Capobianco, F. (2006). My Honest Dual Licensing. Unpublished manuscript, available at www.funambol.com/blog/capo/2006/07/my-honest-dual-licensing.html.

Hecker, F. (1999). Setting Up Shop: the Business of Open-Source Software. *IEEE Software*, 16(1):45–51.

Koski, H. (2007). Private-collective Software Business Models: Cordinatitons and Commercialization via Licensing. Discussion Papers 1091, The Research Institute of the Finnish Economy. Available at http://ideas.repec.org/p/rif/dpaper/1091.html.

Lerner, J. and Tirole, J. (2005). The Scope of Open Source Licensing. *Journal of Law, Economics & Organization*, 21(1):20–56.

Moglen, E. (2003). Free Software Foundation Statement on SCO v. IBM. Document retrieved from www.fsf.org/licensing/sco/sco-v-ibm.html.

Olson, M. (2006). *Dual Licensing.* In DiBona C., Canese C. and Stone M.: Open Sources 2.0: The Continuing Evolution, O'Reilly Media.

Rajala, R., Nissilä, J., and Westerlund, M. (2007). *Revenue Models in the Open Source Software Business.* In Kirk St. Amant and Brian Still eds. Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, IGI Global.

Rosen, L. (2001). Which Open Source License Should I Use for My Software? Open Source Initiative. Document available at www.rosenlaw.com/html/GL5.pdf.

Shapiro, C. and Varian, H. (1998). Versioning: The Smart Way to Sell Information. *Harvard Business Review*, 76(6):106–114.

Välimäki, M. (2005). *The Rise of Open Source Licensing: A Challenge to the Use of Intellectual Property in the Software Industry*. Turre Publishing.

West, J. and Gallagher, S. (2004). Key Challenges of Open Innovation: Lessons from Open Source Software. San José State University, mimeo; document avaiable at www.joelwest.org/Papers/WestGallagher2004.pdf.

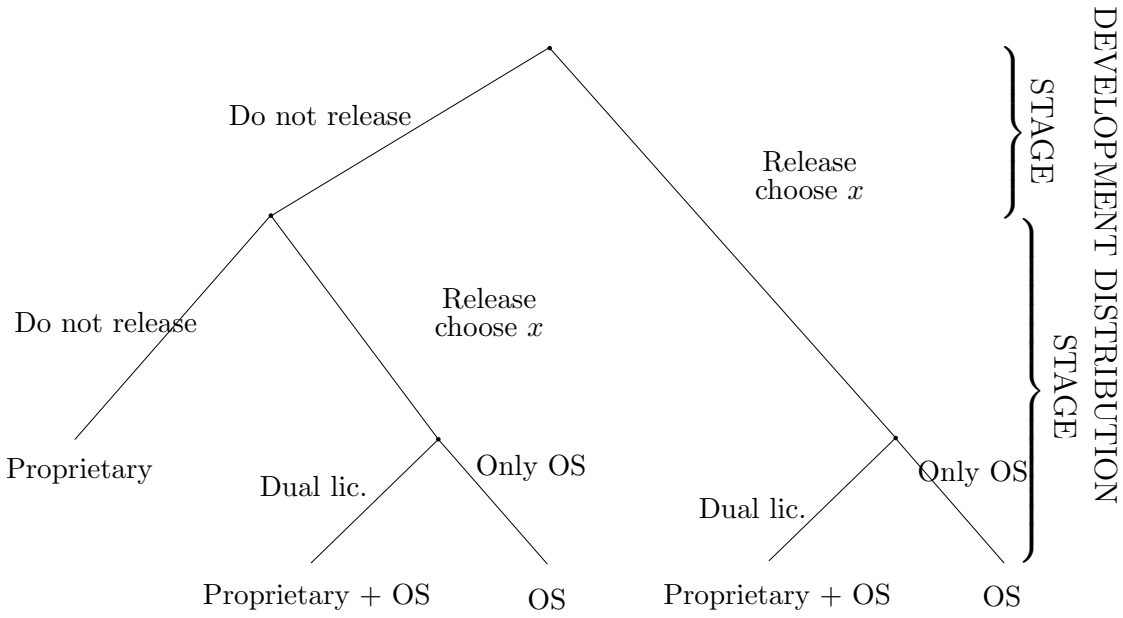

Figure 1: OD decisions mapping