TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

Operations Research and Control Systems

When to Make Proprietary Software Open Source

Jonathan P. Caulkins, Gustav Feichtinger, Dieter Grass, Richard F. Hart, Peter M Kort, Andrea Seidl

Research Report 2011-07

June, 2011

Operations Research and Control Systems
Institute of Mathematical Methods in Economics
Vienna University of Technology

Research Unit ORCOS
Argentinierstraße 8/E105-4,
1040 Vienna, Austria
E-mail: orcos@eos.tuwien.ac.at

When to Make Proprietary Software Open Source

Jonathan P. Caulkins Gustav Feichtinger Dieter Grass
Richard F. Hartl Peter M. Kort Andrea Seidl

June 1, 2011

Abstract

Software can be distributed closed source (proprietary) or open source (developed collaboratively). While a firm loses revenue from sales of a software, the open source software development process can have a substantial positive impact on the quality of a software, its diffusion, and, consequently, the demand for a complementary product. Previous papers have considered the firm's option to release a software under a closed or open source license as a simple once and for all binary choice. We extend this research by allowing for the possibility of keeping software proprietary for some optimally determined finite time period before making it open source.

We show that in case of high in-house R&D costs the firm always makes the software open source at some point (unless switching itself is too expensive). On the other hand, when R&D is inexpensive, the firm opens the source code only when the initial level of software quality is low. For intermediate R&D costs, the firm might even be indifferent between opening the code immediately, open it at some optimally determined time, or keep it closed forever. Finally we find that whereas high switching costs might prevent firms from adopting an open source business model, low switching costs mainly affect the timing of opening the source code.

Keywords: open source; optimal control; multi-stage modeling; complementary product; software

1 Introduction

Firms generating revenue by distributing software face some interesting choices that do not arise for more conventional, physical products. One is whether or not to make the software "open source" in order to tap the talents of volunteer software developers outside the firm in exchange for giving the software away free. Forgoing revenue by giving a product away seems odd, but can make sense if the firm profits from selling complementary products or services whose demand would increase if the software were adopted more widely.

Haruvy et al. (2008) determine how a firm should dynamically optimize both prices and investment in software development over time under both open and closed source conditions. Based on this analysis the firm chooses between these two business models and treats it as a once and for all binary choice. Here, we generalize Haruvy et al. (2008) to allow for the possibility of keeping a product closed source for a finite time before

converting it to open source. The analysis assumes that software quality is the crucial state variable and identifies interesting threshold behavior depending on the initial quality, as well as a variety of other insights. Quality in this context should be understood more generally than merely being free of errors; it also encompasses the existence of features and functionality.

The decision to make a software product open source affects a firm's software development process, its organizational routines, and overall business model. When the source code is freely available, a firm will effectively lose revenues from sales of the software. However, the openness of the code can also have advantages for a firm. As von Hippel and von Krogh (2003) point out, innovation in open source development is typically driven by the users, allowing a software developer to benefit from user contributions with respect to quality, including creativity and security, and, consequently, allowing the firm potentially to reduce in-house software development costs (see also Raymond, 2001; Lerner and Tirole, 2005a).

An open source product also has enormous appeal for customers due to the lack of licensing fees, the possibility to customize the software to their own needs and also to avoid vendor lock-in; see Lerner and Tirole (2002). Thus, making software open will in general increase demand.

Since open source software is mostly given away for free, monetary incentives for a commercial firm to embrace an open source business model come from exploiting the complementaries to one of its commercial products. These complementary products can be hard- or software, of course, or consulting services, maintenance contracts, training, certifications etc. See, e.g., Lerner and Tirole (2002) for a consideration of such business models, but also Economides and Katsamakas (2006); Haruvy et al. (2008); Kort and Zaccour (2011). Examples of firms following such a strategy include Oracle offering training and consulting services for MySQL and Java, and Google selling apps and mobile ads with its (open source) mobile phone operating system Android.

Haruvy et al. (2003, 2008) compare the open source and closed source approach in an optimal control framework. They explore conditions under which it is optimal to release software in open vs. closed source when the firm can make profits by the sales of a complementary product, where switching from closed to open source is not allowed. However, quite a few programs, which are today prominent examples of open source software, were initially released under a proprietary license (e.g., Java, MySQL, Android) or are based to a certain extent on closed source software (e.g., Linux and OpenOffice).

An interesting issue when it comes to making a software product open source is the question of licensing. The basic intention of an open source license is to give the user the right to freely use, access, modify and redistribute the source code (The Open Source Initiative, 2011). One can distinguish between licenses with and without copyleft terms. "With copyleft terms" essentially says that if one re-uses the source code, the derivative product must be operated under the same license terms (see, e.g., de Laat, 2005; Lerner and Tirole, 2005a; Bonaccorsi and Rossi, 2003; see also Mustonen, 2003; Lerner and Tirole, 2005b). The purpose of such restrictive terms is to protect the licensor against the commercial usage of their product by competitors and to give contributing users some kind of assurance that their often un-paid work is not directly commercially exploited. Effectively, this also means that once a software is made open source under a restrictive license, it is not possible to make the software closed source again. (Note, however, that some projects, like Oracle's OpenOffice only accept user contributions if users agree to share or transfer their copyright with the firm, so copyleft restrictions can in fact be

circumvented by a firm in this context.)

Even if making an open source software closed source or restricting access to the product is legally possible, such a decision can still risk adverse effects for a firm, such as loss of reputation, loss of (key) contributors, the formation of so-called forks (the open source code is used for a similar, open project by someone else), etc. A recent example for this can be seen in the open source community's reaction to some of Oracle's unpopular decisions to discontinue its support of several open source project and focus on the proprietary versions of the software.

There can also be switching costs when changing from closed to open source. These costs can either be due to providing communication platforms necessary to give the community a chance to coordinate, or it can be caused by the fact that certain organizational routines might need to be adapted, see e.g., Bonaccorsi et al. (2006).

Since both the open and closed source business models have their advantages, it makes sense to consider the decision whether and when to make a software open source after having it had closed source for a while. Hence, the aim of our paper is to take a closer look at the question of when, if ever, a profit-maximizing firm should switch their business strategy from closed to open source.

We find that the optimal solution is always history-dependent, i.e. it crucially depends on the initial quality, and thus demand, of the software. For certain initial conditions, the optimal strategy is rather clear, e.g. for small initial quality it is optimal to become open immediately and do a little R&D in order to support development efforts by the user community. On the other hand when the initial quality is large, a firm should release its software under a proprietary license, and, depending on parameters like costs for doing in-house R&D development or switching costs, either open it after some optimally determined time or never.

This paper is also related to the literature of history-dependent long-run solutions. There, so-called Skiba points can be detected where the decision maker is indifferent between choosing one of several candidate solutions. Early contributions are Sethi (1977, 1979) and Skiba (1978). A recent exposition can be found in Grass et al. (2008). Concerning the problem studied in this paper, essentially, the firm has to choose between releasing the software open or closed source, and, in the latter case between keeping the source code closed forever or opening it after some time. Unlike in most of the history-dependent long-run solution literature, in this multi-stage framework a firm might not only be indifferent between two solutions that can be distinguished by their different long-term levels of quality (i.e. the state variable), but also between solutions that differ in their long-term business model and the timing of its adoption. This implies that we find indifference (or Skiba) points where the firm has the choice between making the source code open either immediately or never, points where keeping the software closed forever is as good as making it open after a certain time, and points where a firm is indifferent between making the source code open immediately or after some optimally determined time. We will even see that in a hairline case a firm is indifferent about either releasing the software closed and keeping it like this forever, or releasing it closed and open it after some optimally determined time, or releasing it open immediately.

The paper is organized as follows. Section 2 presents the basic model and assumptions. Section 3 contains the analysis, and Section 4 presents the results. Finally, Section 5 concludes.

2 Model Formulation

Let us first consider the classical scenario where a firm wants to sell its software as a proprietary product. As state variable we consider the quality, $K(t)$, of the software. Quality should be understood broadly as the appeal of the product to the consumers. We measure quality by the demand it would generate if the price was zero; in effect it is the intercept of a conventional demand function. The development of the quality level over time satisfies

$$\dot{K} = h(v) - \delta K. \tag{1}$$

The quality level, K, is taken relative to the firm's economic environment in the sense that K only increases when the quality increase of the firm's software exceeds the increase in quality of software of its competitors. This explains the term "$-\delta K$" in (1), where parameter δ denotes the rate of technological obsolescence. The expression $h(v(t))$ measures the impact on quality of investing $v(t) \geq 0$ in R&D. We assume that $h'(v) > 0$ and $h''(v) < 0$ because of diminishing returns.

In the closed source scenario the firm chooses the price of both the software $p_s(t) \geq 0$ and the complementary product $p_a(t) \geq 0$, as well as its research efforts $v(t)$ such that profits are maximized, i.e.

$$\max_{p_s,p_a,v\geq 0} \int_0^\infty e^{-rt}(p_s q_s + p_a q_a - \beta v)dt, \tag{2}$$

where β is the unit R&D costs, r the discount rate, q_s denotes the demand for the software, and q_a the demand for the complementary product. We assume that software demand depends on the quality of the software and its price, i.e.,

$$q_s = K - \varphi p_s, \tag{3}$$

where φ weights the impact of a change in price. Note that our formulation follows Economides and Katsamakas (2006), where "actual sales when all prices are zero" depend on quality, among other things. The demand for the complementary product depends not only on its price but also on demand for the software:

$$q_a = (\alpha - p_a)q_s, \tag{4}$$

where α is a positive constant. Our demand system implies that the complementary product is useless without the software, which makes sense if the software is something basic like an operating system or database, or the complementary product something very specific and adapted to the requirements of the core product, such as training, certifications, consulting, etc. As in Kort and Zaccour (2011), the demand system reflects that the software product and the complementary product are complements, where demand for the the complementary product increases when the price of software decreases. Hence, setting the software price at zero in the open-source case raises demand for the complementary good. This is another argument for the firm to open its source code, in addition to the free increase in quality that is provided by the user network.

In the open source scenario the firm cannot charge a price for the software, thus, the optimization problem becomes

$$\max_{p_a,v\geq 0} \int_0^\infty e^{-rt}(p_a q_a - \beta v)dt, \tag{5}$$

$$\text{s.t.} \qquad \dot{K} = h(v) + g(q_s) - \delta K \tag{6}$$

The function $g(q_s)$ measures the contribution of the open source community to the software quality, and indicates by its dependency on q_s a network effect; the more the software is used, the larger the consumer network and thus the number of potential contributions to raise software quality. In addition, a larger q_s creates a larger incentive (e.g., peer recognition, career concerns, requirements for adaptions; see, e.g., Lerner and Tirole, 2002) to contribute to the project. As contributing programmers often have rather specialized skills, a further increase in demand and users will influence a smaller contributor base; thus, we assume $g'(q_s) > 0$ and $g''(q_s) < 0$.

Unlike Haruvy et al. (2008), we assume that when a firm decides to make a software open source, it keeps control or at least actively continues to support the project, e.g, by taking project leadership, contributing code, etc. (which is something not uncommon for open source projects, see for example Google's involvement regarding Android and Oracle's regarding MySQL and OpenOffice.) Thus, in our model we allow for development costs even if the source code is open.

We employ the specifications $g(q_s) = mq_s^n = mK^n$ and $h(v) = av^b$ with $n, b < 1$, where parameters m and n weight the impact of demand and thus quality (note that $q_s = K$ for $p_s = 0$; see (3)) on the amount of user contribution. The parameters a and b describe the impact of R&D on quality.

When giving the firm the choice of applying closed or open source forever, it turns out that a firm would release its software as open source if the initial quality is low and closed source if the initial quality is high. This result was also obtained in Haruvy et al. (2008). The point where one is indifferent between the two distribution options heavily depends on the parameters, e.g., if R&D efforts are expensive or not efficient, one would exploit the advantages of open source for a larger initial quality range than if development costs were low.

The optimal strategy in the closed source case is to give the software away as freeware when the initial quality is low (i.e., one would distribute the software for free, but keep the source code of the software closed) in order to increase demand of the complementary product. After quality rises through R&D efforts, demand becomes high enough for the firm to charge a positive price. Since both quantity and price turn out to increase proportionally with quality, software revenue has increasing returns in quality. For this reason the incentive to increase quality goes up with the level of quality, which makes that R&D efforts increase as quality goes up.

The optimal strategy for open source software is a little more complicated. One would spend more on R&D than in the closed source case if the initial quality is low to trigger interest of the open source developer community. However, after quality has raised, then quality will be increased by this user network, so that R&D investment by the firm itself is less needed. For this reason, when quality becomes even higher, a firm will perform less in-house development efforts than if the software were sold under a proprietary license.

Now suppose the firm has the option to make a closed source software open after some time. Then the firm optimizes not only the prices and R&D effort, but also determines the optimal time to open the source code. Denoting this switching time as τ, where $\tau \geq 0$, the optimal control problem becomes

$$
\begin{aligned}
\max_{p_s,p_a,v,\tau\geq 0} \quad & \int_0^\tau e^{-rt}(p_s q_s + p_a q_a - \beta v)dt + \int_\tau^\infty e^{-rt}(p_a q_a - \beta v)dt - e^{-r\tau}S, \\
\text{s.t.} \quad & \dot{K} = h(v) - \delta K \qquad \text{for } 0 \leq t < \tau, \\
& \dot{K} = h(v) + g(K) - \delta K \qquad \text{for } \tau < t \leq \infty, \\
& S = \begin{cases} S_0 & \text{if } \tau = 0, \\ S_\tau & \text{if } \tau > 0, \end{cases} \\
& K(0) = K_0, \\
& p_s = 0 \text{ for } \tau < t \leq \infty.
\end{aligned}
$$

Parameter S denotes the switching costs incurred when making a proprietary software open source. We allow for the switching costs S_0 at $\tau = 0$ to be different than the switching costs S_τ at $\tau > 0$. Bonaccorsi et al. (2006) argues that costs associated with switching from closed to open source are higher, the more established the software is. Hence, switching costs may be lower at the start of the planning period compared to later on when the business model and processes are already implemented. Consequently, we assume $0 \leq S_0 \leq S_\tau$. Also, one can interpret S_0 as the difference in costs for initially releasing the software either closed or open source.

In this framework we only allow switches from closed to open source. This is motivated by the fact that in practice it hardly occurs that open source software is made closed. There are several reasons for this like legal issues concerning copyright and copyleft restrictions of user contributions and the danger of offending the developer community leading to reputation losses. Note that this assumption implies that the firm can change its strategy with respect to open or closed source licensing at most once.

3 Analysis of the Model

We approach the problem in a two-stage model with τ being the switching time. Then, $\tau = 0$ essentially means the firm applies open source forever, $\tau \to \infty$ implies closed source forever, while a finite positive τ means the firm starts with closed source, but switches to open source at time τ. The Hamiltonian for the first stage (closed source) is

$$\mathcal{H}_c = p_s(K - \varphi p_s) + p_a(\alpha - p_a)(K - \varphi p_s) - \beta v + \lambda_c(a v^b - \delta K),$$

and for the second, open source, stage it is

$$\mathcal{H}_o = p_a(\alpha - p_a)K - \beta v + \lambda_o(a v^b + m K^n - \delta K),$$

where $\lambda_i(t)$, $i = c, o$ denotes the costate variable, for which it always holds that $\lambda_i \geq 0$; see Leonard (1981).

Since control constraints can play a role, we also have to consider the Lagrangian functions

$$\mathcal{L}_c = \mathcal{H}_c + \mu_1 p_s + \mu_2 p_a + \mu_3 v, \qquad \mathcal{L}_o = \mathcal{H}_o + \mu_2 p_a + \mu_3 v.$$

By applying Pontryagin's maximum principle, we find in the interior of the admissible region that

$$p_s^* = \frac{1}{2}\left(\frac{K}{\varphi} - \frac{\alpha^2}{4}\right), \qquad p_a^* = \frac{\alpha}{2}, \qquad v^* = \left(\frac{ab\lambda}{\beta}\right)^{\frac{1}{1-b}}$$

for the closed source stage and

$$p_a^* = \frac{\alpha}{2}, \qquad v^* = \left(\frac{ab\lambda}{\beta}\right)^{\frac{1}{1-b}},$$

for the open source stage, where due to the model assumption, the price of the software equals zero, i.e. $p_s = 0$. The Legendre-Clebsch condition is fulfilled since the costate is non-negative, see, e.g., Grass et al. (2008). Also, the control constraint concerning v will never become active, so that it always holds that $\mu_3 = 0$. Further, we see that the price for the complementary product is always positive and constant, which implies that $\mu_2 = 0$. The control constraint concerning the non-negativity of the price of the software becomes active when $K \leq \frac{\alpha^2\varphi}{4}$. The corresponding Lagrange multiplier can be determined to be

$$\mu_1 = \frac{\alpha^2\varphi}{4} - K.$$

Note that by Pontryagin's maximum principle we can also identify other price levels for the closed source stage fulfilling the necessary conditions for the price of the software and the complementary product, i.e. $p_s^* = \frac{K}{\varphi}$ and $p_a^* = \frac{\alpha\varphi \pm \sqrt{(\alpha^2\varphi+4K)\varphi}}{2\varphi}$. However, we can exclude that these prices are optimal, because the resulting demand for the software and complementary product become zero, and, thus, profits will always be zero. It can easily be shown that the value of the Hamiltonian for this set of prices is always below the one for the other set of prices.

The costate equations for the two stages are

$$\begin{aligned} \dot{\lambda}_c &= (r+\delta)\lambda_c - p_s - p_a(\alpha - p_a), \\ \dot{\lambda}_o &= (r+\delta)\lambda_o - p_a(\alpha - p_a) - \lambda_o mnK^{n-1}. \end{aligned}$$

At the optimal switching point, reached at time $\tau^* \geq 0$, the matching conditions (see, e.g. Tomiyama and Rossana, 1989; Makris, 2001)

$$\mathcal{H}_c(\tau^*) + rS = \mathcal{H}_o(\tau^*) \qquad \text{and} \qquad \lambda_c(\tau^*) = \lambda_o(\tau^*) \tag{7}$$

have to be fulfilled. One can also rewrite these conditions as

$$\lambda_i(\tau^*) = -\frac{1}{m}K(\tau^*)^{-n}\left(\frac{1}{8}\alpha^2 K(\tau^*) - \frac{1}{4}\frac{K(\tau^*)^2}{\varphi} - \frac{1}{64}\alpha^4\varphi - rS\right) \qquad \text{for} \quad i = c, o. \tag{8}$$

4 Making a Software Product Open Source

From Haruvy et al. (2008) we already know that the decision whether to release a software closed or open source depends crucially on the initial quality. But what happens if we allow the firm to open the source code at some future point in time after starting out with closed source?

Let us first assume that there are no switching costs, i.e. $S = 0$, in order to gain some basic understanding of the model. We chose the parameters presented in Table 1, with the in-house R&D cost parameter β treated as a bifurcation parameter.

If the costs of doing R&D are low, i.e. $\beta < 7.977$, the results are the same as in the Haruvy et al. (2008) paper. If the initial quality is low, it is optimal to release the software open source. On the other hand, if the initial quality is high it is optimal to

r	α	φ	δ	a	b	m	n
0.1	5	5	0.05	1	0.5	0.25	0.5

Table 1: Parameters used for the numerical calculations.

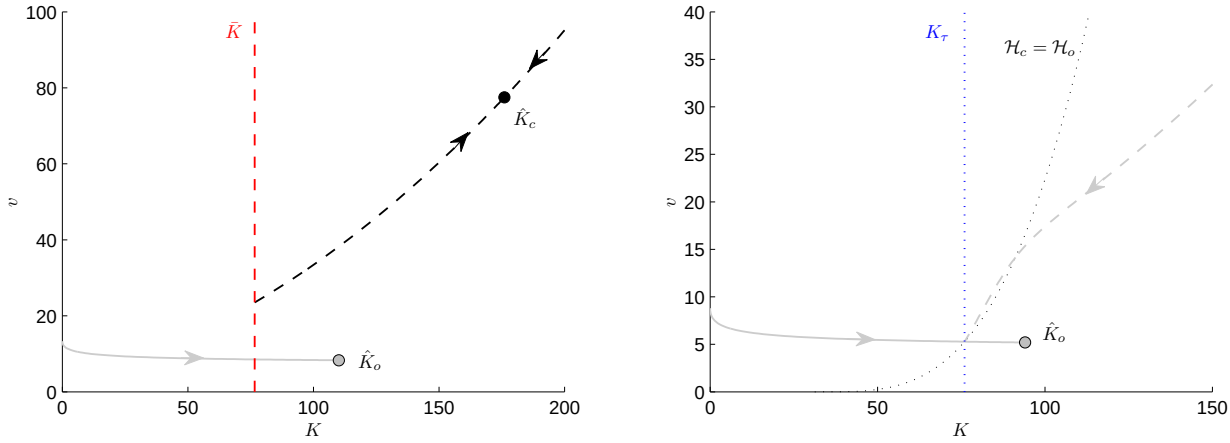

Figure 1: Phase portrait for low and high R&D costs ($\beta = 7.85$, left panel, and $\beta = 10$, right panel). The gray line shows the optimal solution path leading to the open source steady state, while the black line depicts the solution path leading to the closed source steady state $\hat{K}_c$. A solid line means on this part of the solution open source is optimal, while a dashed line means closed source is. In the left panel $\bar{K}$ denotes an indifference point and K_τ is an optimal switching point determined by the matching conditions (7) in the right panel; left to this point an open source release is optimal, right to this point a closed source release is.

release the software closed source and keep it this way. At a certain level of quality, $\bar{K}$, one is indifferent between an open and closed source release. In the optimal control literature, $\bar{K}$ is called a Skiba point (see Sethi, 1977, 1979; Skiba, 1978; Grass et al., 2008). In this scenario, no matter in which stage one starts, at $K_0 = \bar{K}$ quality would still increase as can be seen in the left panel of Figure 1.

As an aside, Figure 1 suggests an interesting avenue for further research. In case of open source it is optimal to increase quality, but conduct R&D efforts that slightly decrease over time, until a quality level $\hat{K}_o$ is reached. Then the firm is in a steady state, so that quality is kept constant from there on. However, when the initial level of quality is $K_0 = \hat{K}_o$, it is not optimal to release the software open source and remain in that steady state. One would prefer instead to approach the closed source steady state by keeping the software under a proprietary license and spending more on R&D. Yet, in this model the open source steady state is optimal in the long term if starting with small initial quality. The reason for this is the modeling assumption that once the source code is open, it is not possible to make it closed again. So, if a firm had the option to close software that had been open, it could be optimal to do so at some optimally determined time if the initial quality is low. It is common to assume that copyleft provisions or other considerations make transitions one way; so once open, always open. There are some exceptions, however, so a model that allows switching in either direction could be worth investigating in future work.

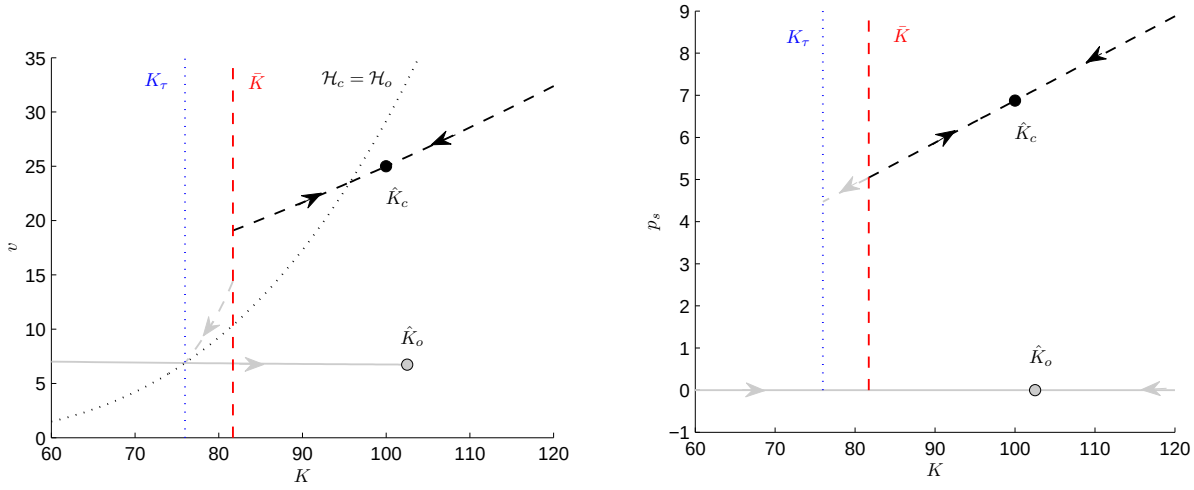

Figure 2: Phase portrait for intermediate value of β. ($\beta = 8.75$) The left panel shows the optimal strategy with respect to development efforts while the right panel shows the optimal pricing strategy for the software.

On the other hand, if costs for R&D are large, i.e. parameter $\beta > 9.066$, it is too expensive to keep quality high in the long run using only in-house development. Of course, when the initial quality is large software demand is large so that it pays to charge a positive price for software. Therefore, it is still optimal to initially release the software as closed source, but due to the high costs, less effort is invested on R&D and so quality falls, as in the right panel of Figure 1. At the optimally determined time τ^*, quality is so low that it becomes optimal to make the software open source and take advantage of the user community. Instantaneously, demand for the software and the complementary product increases since the price for the software becomes zero. On the other hand, if the initial quality, and thus software demand, is low, one would release the software open source in order to push sales of the complementary product. We conclude that for large β the firm would always open the source code at some point; the only question is when.

If parameter β takes some intermediate value, Figure 2 shows that no threshold exists at which the decision maker is indifferent between releasing the source code closed or open source. But there is still a Skiba point. Starting exactly at this point, $\bar{K}$, it is always optimal to release the software closed source. Yet, the firm is indifferent between starting to increase R&D efforts and keep the source code closed forever, and starting to decrease R&D efforts until quality level K_τ is reached at time τ^*. In the first case, quality will grow and the firm can make profits by selling the software as well as the complementary product, until one reaches a steady state value. This policy is optimal whenever the initial quality exceeds $\bar{K}$. In the second case, due to the lower R&D efforts, quality will fall, so one would charge a lower price for the software. After some time τ^* it becomes optimal to make the software open source. The user community will grow since the software is available for free then and more people will start to contribute to the source code and quality of the software. Then quality can grow again until it reaches its open source steady state value. If the initial quality falls below K_τ, it is optimal to release the software open source or make it open immediately. If the initial quality is very high it is optimal to release the software closed source and keep it this way. Note that again the exclusion of the possibility of making the software closed once it is open

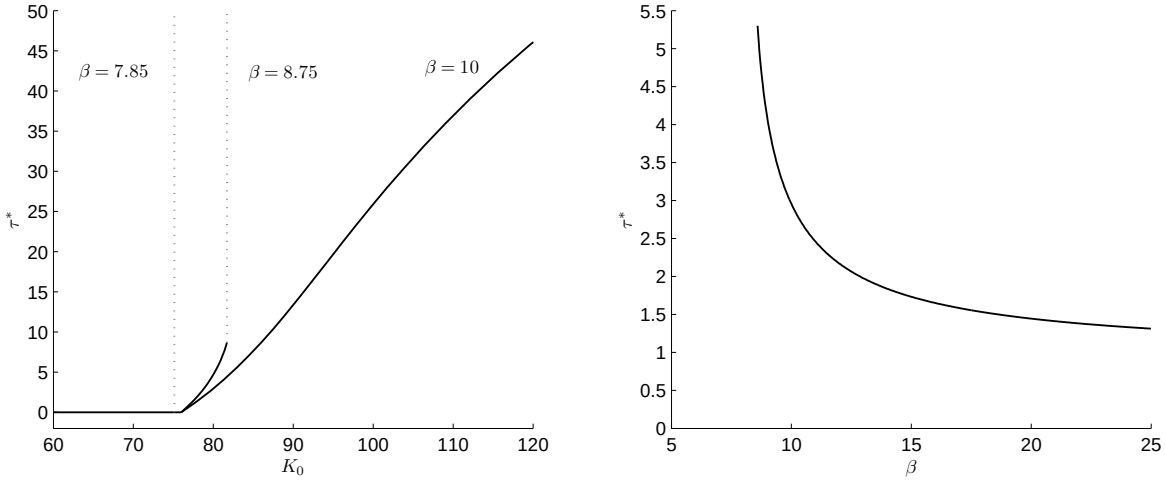

Figure 3: The optimal switching time depending on the initial quality K_0 and parameter β. In the right panel K_0 is fixed at 80.

plays an important role with respect to the monotonicity and optimality of the solution path. In particular, if one starts to move left (reduce the state from $\bar{K}$) one can end up further to the right than if one had initially decided to move to the right, which of course could not happen in a one-stage, one-state model.

The right panel in Figure 2 shows the optimal strategy with respect to pricing. The higher the quality of the proprietary software is, the higher its price. Since the optimal price depends only on K and some parameters, there is no discontinuity at the indifference point as there was for the development efforts. Yet, depending on the chosen strategy, the decision maker would either increase or decrease the price related to the corresponding quality.

A few remarks are in order on the transition between the three parameter regions. At $\beta = 7.977$, K_τ, the point on the solution path leading to the open source steady state where the matching conditions (7) are fulfilled, coincides with the Skiba point, $\bar{K}$, where the firm is indifferent between open and closed source release. For smaller β, a point K_τ also exists, but, although the matching conditions are fulfilled at this point, it is never optimal to actually switch from closed to open source. At $\beta = 9.066$ we can find a heteroclinic bifurcation, i.e. (the closed source part of) the stable manifold of the open source steady state $\hat{K}_o$ and the unstable manifold of the closed source steady state $\hat{K}_c$ coincide. In this hairline case this means starting with $K_0 < \hat{K}_c$ the firm always opens the source code after some optimally determined time. If on the other hand $K_0 \geq \hat{K}_c$ the firm keeps the software closed forever. This kind of bifurcation is often related to the occurrence/disappearance of multiple optimal solutions: While for β larger than this bifurcation value the firm always optimally ends up at the open source steady state, for smaller β the long-term outcome depends on the initial state value and at a certain point the firm is indifferent between finally ending up in the open and in the closed source steady state.

4.1 Optimal Switching Time

In the previous section we saw that the larger the initial quality and, thus, demand are, the longer is the solution path until the firm switches from closed to open source. The

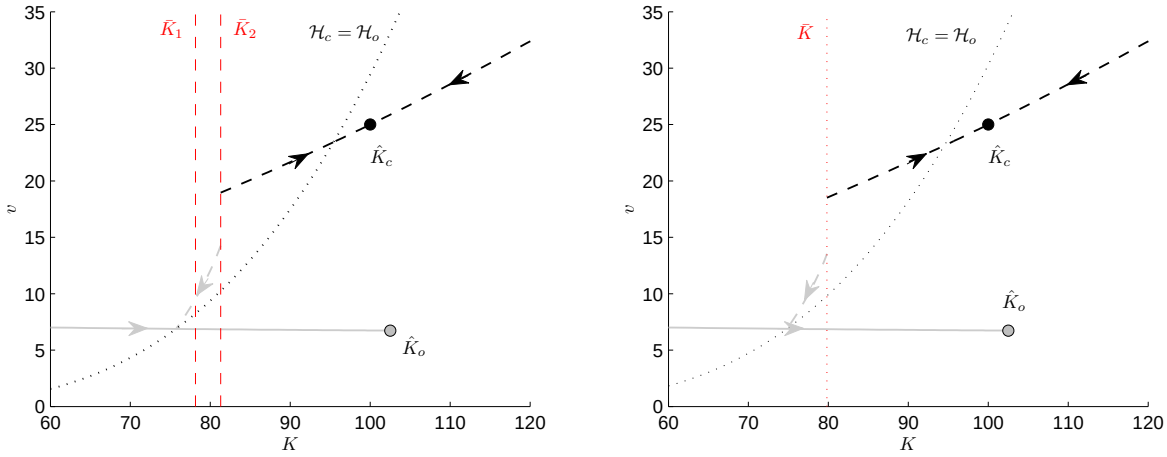

Figure 4: Phase portrait for $\beta = 8.75$, switching costs $S_\tau = 10$ (left panel) and $S_\tau = 44.421$ (right panel), and $S_0 = 0$.

left panel of Figure 3 shows the optimal switching time τ^* for different initial state values. While for low R&D costs it can be seen that the firm either switches immediately if the initial quality is low or never (i.e. S_0 is on the right side of the corresponding dotted line), for a high β it is always optimal to make the software open source. The larger the initial quality, the longer the firm should wait to open its code.

The optimal switching time depends not only on the initial state, but also on the different parameters. An increase in parameters m, α, β, δ or φ reduces the optimal switching time by making open source more attractive relative to the closed source stage. A higher α means that more complementary products can be sold (which is of course easier when the primary software is given away for free). A higher β (see the right panel of Figure 3) implies that in-house R&D becomes more expensive, so that applying open source becomes more attractive, since in the latter case software quality can also be increased by the consumers' network. Increasing δ makes it more difficult to keep quality up with the technological progress in the firm's economic environment, and being in open source helps to counter this effect. And when customers are less willing to accept a higher software price, i.e. φ is higher, less profits can be made from closed source sales and thus, open source becomes attractive sooner.

However, increasing the parameter a delays the optimal switching time, because R&D is more efficient, so quality can be increased more easily even without the help of the open source community.

4.2 Switching Costs

So far it was assumed that making software open source imposes no costs on the firm. This, however, might not be entirely realistic (see, e.g., Bonaccorsi et al., 2006). For this reason we introduce switching costs.

First, consider the case where there is no cost when releasing the software as open source from the outset, so $S_0 = 0$, but there are costs S_τ for converting the software to open source at some later time $\tau > 0$. Obviously, when R&D costs are small, nothing changes as it is not optimal to change one's licensing strategy anyway.

However, when it is optimal to change the code release strategy after some time,

switching costs do affect the optimal solution. At the switching point costs are incurred, making solutions with switching less advantageous. This implies that for quality levels a little higher than the one corresponding to the switching point, it is better to release the software immediately as open source. However, when switching costs are not too high, it might still pay off for some larger initial quality level to release the code first closed and make it open afterwards. Then we can find an indifference point where the firm has the choice between releasing the software closed source and making it open after some optimally determined time, and releasing it under an open source license immediately.

Another effect of switching costs is that the switching curve (8) shifts upwards. Consequently, the quality at the switching point will be lower compared to the scenario with no switching costs. For high initial quality this means one would wait longer until changing the strategy; i.e., since it is more expensive, the firm delays the moment at which the source code is opened. Note, however, that a firm would only open the code later, if it is optimal to make the software open for no switching costs, and that high switching costs might prevent a firm from embracing open source at all.

In the case of intermediate R&D costs, switching costs create the situation shown in Figure 4. If initial quality is low, releasing software open source is optimal. If quality has some intermediate initial level, the optimal strategy is to release it closed source and make it open after some optimally determined time. For the initial quality $\bar{K}_1$ one is indifferent between these two options. If quality is large, it is optimal to release the software closed source and keep it like this. $\bar{K}_2$ denotes a Skiba point, where one would always initially release the software closed source, but still be indifferent between opening it up at some optimally determined time and keep it closed forever.

If switching costs increase further, it gets less and less advantageous to change an established strategy. Thus, $\bar{K}_1$ shifts to the right and $\bar{K}_2$ to the left, until they coincide. In this hairline case, which can be seen in the right panel of Figure 4, there is only one indifference point, but there the firm has three choices: release the software open and keep it open, release it closed and make it open after some time, and release it closed and keep it closed.

If the switching costs are even higher, it is not optimal to change a strategy once chosen; it simply is too expensive. Yet, we still have the indifference point where a decision maker can choose between an open or closed source release. This is in fact the situation analyzed by Haruvy et al. (2008).

If the in-house R&D costs are high, we also find an indifference point for low switching costs, as shown in the left panel of Figure 5. There one has the choice between (1) releasing software open source and keeping it this way and (2) releasing it closed source and making it open after some optimally determined time. The first (second) strategy is otherwise optimal for an initial quality level being low (high). When switching costs get higher, switching becomes less attractive, implying that the quality level becomes higher at the indifference point.

Due to the upward shift of the switching curve (8), we have a heteroclinic bifurcation again for a certain level of switching costs, as can be seen in the right panel of Figure 5. The unstable manifold of the closed source steady state $\hat{K}_c$ and the (closed source part of the) stable manifold of the open source steady state $\hat{K}_o$ coincide. We can see in Figure 5 that it is not possible anymore to start at $K_0 = \bar{K}$ and optimally end at the open source steady state, but the closed source steady state starts to play a role again: If switching costs reach or exceed this bifurcation level, it is again optimal not to switch at all for $t > 0$; only the release decision is affected by the quality. Note that for high

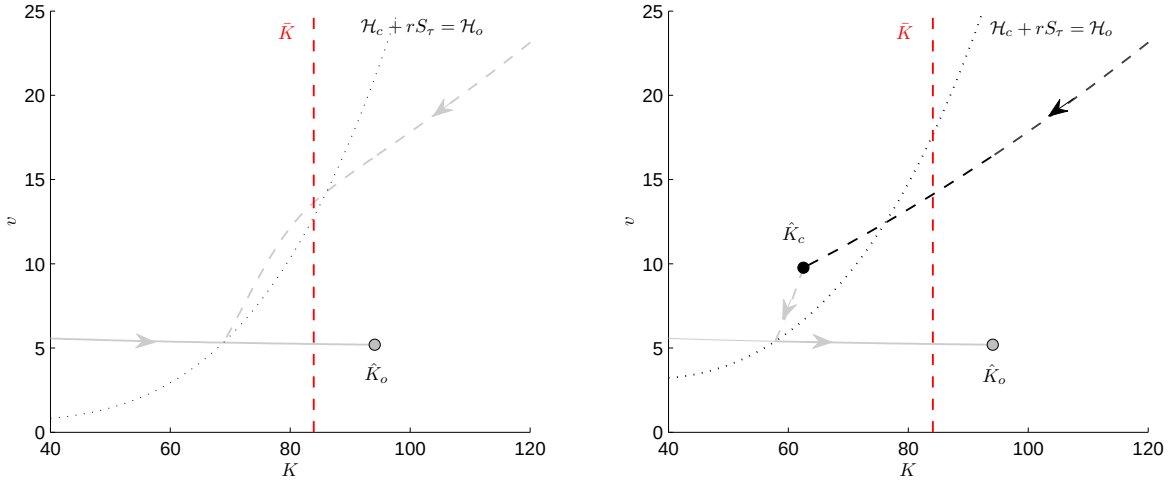

Figure 5: Phase portrait for $\beta = 10$, switching costs $S_\tau = 250$ (left panel) and $S_\tau = 530.586$ (right panel), respectively and $S_0 = 0$.

R&D costs, the level of the switching costs, where one would stop to switch at $\tau^* > 0$, is much higher compared to the scenario where development costs are low. This makes sense as the incentives for switching are higher if β is large.

Figure 6 shows how the optimal switching time depends on the initial state value and on the switching costs. We can see that for high switching costs the firm should - depending on the initial quality - either wait longer until opening the source code or do it immediately. Note that the discontinuity of the optimal switching time is due to the indifference points, where the firm has the choice between making the source code open either after some optimally determined time and immediately or never.

Let us consider now what happens if we have some costs for initially releasing the software open source, i.e. $0 < S_0 \leq S_\tau$. Obviously, solutions, where one would switch immediately from closed to open source become less advantageous. However, costs for initially releasing the software open neither have an impact on the location of switching points (determined by the matching conditions (7)), nor on points where one is indifferent between keeping a software closed forever and making it open at time $\tau^* > 0$. Points where one is indifferent between releasing it open source and releasing it closed source occur for lower levels of quality. So for high initial switching costs it might not be optimal anymore to release the software open source even if the initial quality is rather low. For intermediate and high development costs and low switching costs S_τ, the indifference point between open source release and closed source release with making the code open at time τ^* shifts to the left and disappears for $S_0 = S_\tau$ as it coincides with the optimal switching point K_τ. The reason for this impact of initial switching cost is that it simply shifts the value of the objective function downward for solutions where one would immediately release the software open source, but does not affect the other solutions.

5 Conclusion & Extensions

We saw that the decision whether to distribute software closed source or open source may not be a simple once and for all binary choice, but crucially depends on the used parameters. However, no matter how high the costs for R&D or switching are, the initial

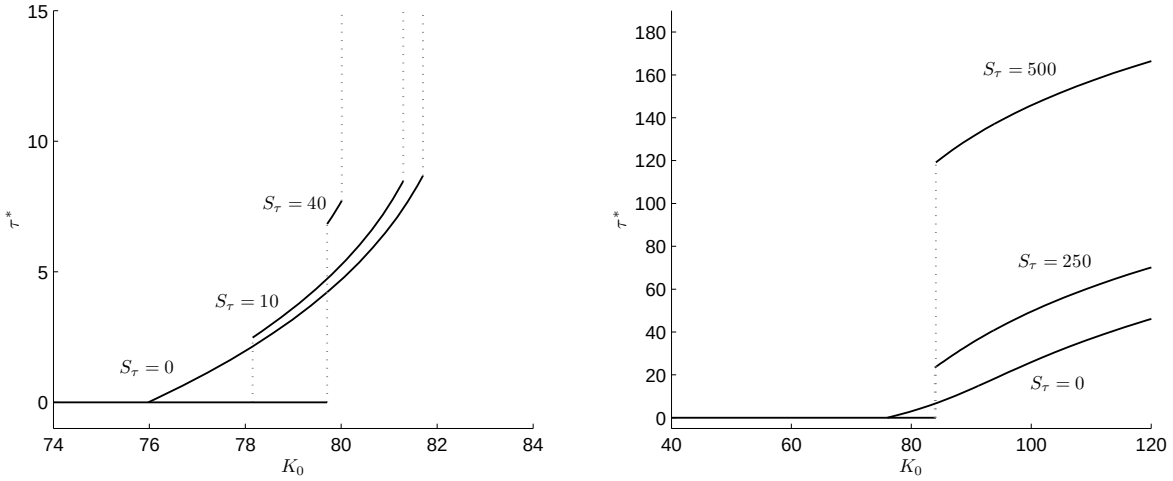

Figure 6: The optimal switching time depending on the initial quality K_0 and switching costs S_τ for $\beta = 8.75$ (left panel) and $\beta = 10$ (right panel).

conditions always play an important role for answering the question which business model is optimal. When the initial quality is low, a firm might want to exploit contributions by the open source community, while for a high initial quality a firm would prefer at least for some time to make money by selling the software.

We found that when it is cheap to do in-house software development, the firm would open the source code only if the initial quality is very low. On the other hand, if R&D costs are high the firm would always make the software open source, but for a large initial quality it would first make some money by selling the software. For intermediate R&D costs, the optimal strategy depends even more on the initial quality level. For a very small initial quality it is optimal to make the software open source immediately, for large quality one would keep it closed. In between, the optimal strategy is to release the software closed source and make it open at some optimally determined time. At a certain point a firm is even indifferent between keeping the software under a proprietary license forever and making it open at some optimally determined time.

Of course, the different parameters have a substantial influence on the question of whether and when to make a source code open. Particularly switching costs can have a large impact on the optimal strategy in general as well as on the timing of making the source code open. While it is not optimal at all to change a strategy once chosen when switching costs are high, under low switching costs the range of quality where one would release the software open source immediately as well as where one would never open the code become smaller. We can find two indifference points, where a firm has the choice between either releasing the software open source immediately or never and between never and releasing it after some optimally determined, finite time. For a certain level of switching costs these two indifference points coincide and the firm becomes indifferent between all three options there.

An important assumption in this paper was that once the source code is made open, it is impossible to make it closed again. Yet, there are circumstances where the firm would like to switch from open to closed if that was possible. For instance, it would make sense that for small initial quality a firm takes advantage of the user community first and when quality is sufficiently high makes the software closed. However, Landini (2011) showed

that in practice only a very small fraction of firms really take this path.

It can be shown that if we omit the restriction with respect to the switching direction, it might in fact even be optimal to switch back and forth between the two stages with optimal switching time zero. Introducing switching costs in this scenario can lead to the occurrence of a cyclical solution. In such a scenario we also find that limiting the maximum number of switches can have a substantial impact on the optimal solution. While such solutions might be of little relevance for a decision maker in terms of being realistic in an open-closed source context, they might deliver interesting insights into multi-stage models and be of interest for different application fields.

Acknowledgements

This research was supported by the Austrian Science Fund (FWF) under Grant P21410-G16.

References

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), 1085–1098.

Bonaccorsi, A. and Rossi, C. 2003. Why Open Source software can succeed. *Research Policy*, 32 (7), 1243–1258.

de Laat, P. B. 2005. Copyright or copyleft? An analysis of property regimes for software development. *Research Policy*, 34 (10), 1511–1532.

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

Grass, D., Caulkins, J. P., Feichtinger, G., Tragler, G., and Behrens, D. A. 2008. *Optimal Control of Nonlinear Processes: With Applications in Drugs, Corruption and Terror*. Springer, Heidelberg.

Haruvy, E., Prasad, A., and Sethi, S. P. 2003. Harvesting altruism in open-source software development. *Journal of Optimization Theory and Applications*, 118 (2), 381–416.

Haruvy, E., Sethi, S. P., and Zhou, J. 2008. Open source development with a commercial complementary product or service. *Production and Operations Management*, 17 (1), 29–43.

Kort, P. M. and Zaccour, G. 2011. When should a firm open its source code: a strategic analysis. to appear in Production and Operations Management, DOI: 10.1111/j.1937-5956.2011.01233.x.

Landini, F. 2011. Technology, property rights and organizational equilibria: An explanation of the co-existence of open and closed source productions. *Empirical Studies of Firms & Markets eJournal*, 31 (3).

Leonard, D. 1981. The signs of the co-state variables and sufficiency conditions in a class of optimal control problems. *Economic Letters*, 8, 321–325.

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

Lerner, J. and Tirole, J. 2005a. The economics of technology sharing: open source and beyond. *Journal of Economic Perspectives*, 19 (2), 99–120.

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

Makris, M. 2001. Necessary conditions for infinite-horizon discounted two-stage optimal control problems. *Journal of Economic Dynamics and Control*, 25 (12), 1935–1950.

Mustonen, M. 2003. Copyleft – the economics of linux and other open source software. *Information Economics and Policy*, 15, 99–121.

Raymond, E. S. 2001. *The Cathedral and the Bazar*. O'Reilly Media, Sebastopol, CA, USA.

Sethi, S. P. 1977. Nearest feasible paths in optimal control problems: Theory, examples, and counterexamples. *Journal of Optimization Theory and Applications*, 23 (4), 563–579.

Sethi, S. P. 1979. Optimal advertising policy with the contagion model. *Journal of Optimization Theory and Applications*, 29 (4), 615–627.

Skiba, A. K. 1978. Optimal growth with a convex-concave production function. *Econometrica*, 46 (3), 527–539.

The Open Source Initiative 2011. The open source definition. http://www.opensource.org/docs/osd. (last accessed Feb, 28th, 2011).

Tomiyama, K. and Rossana, R. J. 1989. Two-stage optimal control problems with an explicit switch point dependence: Optimality criteria and an example of delivery lags and investment. *Journal of Economic Dynamics and Control*, 13 (3), 319–337.

von Hippel, E. and von Krogh, G. 2003. Open source software and the "private-collective" innovation model: Issues for organization science. *Organization Science*, 14 (2), 209–223.