Open Source Software and
Network Externalities

Gerhard Clemenz
University of Vienna
Department of Economics
BWZ, Bruenner Strasse 72, 1210 Vienna, Austria

First Draft: February 2010
This Draft: June 2010

Abstract: It is investigated whether there exist conditions under which (i) only open
source software (OSS) is viable but not commercial software (CS); (ii) OSS is
developed more efficiently than CS; (iii) OSS would be of higher quality than CS; (iv)
market entry is possible for OSS but not for CS. The answer is in the affirmative. A
key assumption concerns the production function for software which exhibits
economies of collaboration, but also diseconomies of scale if the number of
developers gets too large. Heterogeneity of users and the existence of a small, highly
motivated group are conducive for OSS.

JEL classification: L17, L23,

Keywords: Open source, software, private provision of public good.

Acknowledgement: I would like to thank Sandro Shelagia, Dongsoo Shin, Oz Shy
and Konrad Stahl for useful comments on an earlier draft, the usual disclaimer
applies.
1. Introduction

Open source software (OSS) is often considered to be a viable – if not superior – alternative to commercial or closed software (CS). Proponents of OSS frequently claim that

- OSS can be developed under circumstances which would not permit the profitable development of CS even though the software would yield a social net benefit.
- OSS is developed more efficiently than CS.
- OSS would be of higher quality than CS.
- OSS can enter a market and break a monopoly under conditions which would not allow profitable entry for CS.

Some Gurus of the OS-community even claim that OS offers a new way of organizing production and distribution which is superior to traditional ones like firms and markets, and would be viable far beyond the software industry (Benkler 2001). While such claims seem a bit exaggerated it is a fact that OS-projects have been quite successful over the last 15 years and look quite likely to continue to do so. The purpose of this paper is to investigate whether there exist plausible conditions under which one or more of the above claims hold true.

Starting point of the analysis are the models of Johnson (2002) and Bitzer et al. (2007) who explain the development of OSS as private provision of a public good. Adding to these models a positive consumption externality shows that it is indeed possible that OSS may be provided whereas CS would not be profitable. The reason is that for OSS-developers the software yields direct utility whereas for the developers of CS the only benefits are the returns from sales. On the other hand, for the provider of OSS it
is efficient to give away the software for free in order to ensure that he can enjoy the positive network externality.

A shortcoming of this approach is that OSS is developed just by one potential user. The most striking feature of OSS, however, is the collaboration of many developers who are not paid for their contributions. In order to capture this feature we assume that there also exists a network effect in the production of software. In particular, we assume a production function with labor hours as only input. The number of labor hours required to produce software of given quality depends on the number of programmers involved. Total labor input is at first decreasing in the number of programmers due to positive network effects, i.e. there exist “economies of collaboration”. If their number exceeds a critical value, co-ordination problems start to dominate and the required labor input starts to grow. Potential users differ with respect to the utility obtained from the software. A necessary condition for OSS to be developed is that the sum of the maximum labor inputs the potential users with the highest valuation of the software are willing to contribute suffices to create the software before the diseconomies of scale become too large. As far as CS is concerned we assume that the costs of hiring developers – i.e. advertising, screening, training etc. – are higher than for OSS where there is appropriate self selection. Under these assumptions OSS may require less total programming time and may be produced cheaper than CS. But even if this is not the case it is still possible that OSS could be produced under conditions under which CS would not be profitable. On the other hand, however, diseconomies of scale and too little willingness to contribute may prevent OSS from getting started whereas CS could be provided. The intuition behind this result is that a commercial software firm cannot capture the entire consumer surplus and thus tends to under provide software. Externalities are a second
source of market failure. OSS-projects can overcome these problems, but they may suffer from diseconomies of scale in software production.

Similar reasoning can be applied to the question whether OSS or CS would be of higher quality. It can be shown that conditions exist under which OSS would have higher quality than CS. A potential problem for OSS is that the fraction of potential users with a high valuation of quality must be sufficiently large, otherwise a critical mass is not reached.

Finally, it is shown that OSS may successfully challenge a commercial incumbent if it is of higher quality than the established CS, even if alternative CS of the same quality could not profitably enter the market.

To sum up, for all the claims stated above there exist conditions under which they are true, but OSS is not superior under all possible circumstances. Given empirical evidence, this is as to be expected as we observe co-existence of OSS and CS. It is noteworthy that it is not assumed that contributors to OSS enjoy additional benefits like signaling their quality, reputation in the community, satisfaction from a cool project or breaking a monopoly. Taking these into account would of course strengthen the case for OSS even further.

The paper is organized as follows. In the next section we review briefly the literature on OSS which is related to our analysis. In section 3 we present a simple formal model of the private provision of a public good approach proposed by Johnson (2002) and Bitzer et al. (2007). In section 4 we introduce positive network externalities occurring for the users of software and demonstrate how this can help to explain why in some situation OSS can be viable whereas CS would make losses. Then we introduce a cost function for the development of software which takes both, economies of collaboration and diseconomies of scale into account. We discuss
circumstances which favor either CS or OSS. We apply this approach to the question whether OSS may produce higher quality than OS, and whether OSS may successfully break a monopoly where CS would fail to do so. We conclude by discussing our results and suggesting areas of future research.

2. Related Literature

According to some economists “Open source software is an economic paradox” (Fershtman and Gandal 2007, p.209), because it is often developed by unpaid volunteers and given away for free, including the source code. Consequently, much of the literature is focused on two questions: what motivates the contributors to participate in an OSS-project, and why and under which circumstances is software made freely available to the public. A natural follow up question, of course, concerns the coexistence of OSS and CS. In the present paper we are concerned with the question of licensing insofar as a license like GNU allows an initiator of an OSS project to enjoy network externalities both in the use and in the development of the software, but we take the free availability of OSS as given and do not dwell on that question in detail.¹ Our focus is the incentive to participate in the development of OSS and its viability.

The fact that many contributors to OS-projects are not paid for their work has led economists to suggest a number of non-pecuniary benefits and future money returns derived from participation in a project (see e.g. Lerner and Tirole 2002, 2005a). Non-pecuniary benefits are the intrinsic pleasure of participating in a “cool” project, ego gratification from peer recognition (“egoboo” according to Raymond 2000), a desire of fighting monopolies, or plain altruism. Future monetary benefits may occur because of improvement of skills, better standing in the job market through

¹ For a detailed analysis of OS licensing see Lerner and Tirole (2005b).
“signaling” of abilities, learning more about one’s own abilities, acquiring shares in commercial OSS-based companies or easier future access to venture capital markets. Subsequent empirical research, some of it based on questionnaires (e.g. Ghosh et al. 2005, Lakhani and Wolf 2005) some using econometric studies (e.g. Fershtman and Gandal 2007, Sauer 2007) have indeed confirmed the importance of these motives. Many well known OSS projects, however, have been initiated by users who for one reason or other were dissatisfied with the available CS. Examples are Apache (started by B. Behlendorf); Linux (Linus Torvalds); Perl (Larry); Sendmail (Eric Allman) etc. For projects like these an alternative approach, based on the literature on the private provision of a public good, seems fairly well suited. This route has been taken by Johnson (2002), Bitzer et al. (2007) and Myatt and Wallace (2008) (see section 3 below). A main shortcoming of these models is that in equilibrium the OSS is developed by just one contributor. The present model amends this approach by introducing network effects and analyzing the collaboration of potential users directly.

As far as competition between OSS and CS is concerned the present paper is restricted to the question under which conditions OSS may successfully enter a market with CS enjoying a monopoly position. It is left to future research to analyze in more details competition between OSS and CS, an area which is still largely unexplored, with Lanzi (2009) being a notable recent exception.

---

2 As the source code is not passed on to other users the model is more apt to explain the creation of freeware rather than OSS. In what follows we shall neglect this distinction.

3 His model analyzes two firms with different objective functions, the characteristics of OSS are not really captured.
3. The public good approach

Some authors have viewed open source software projects as analogous to the provision of a public good (Bitzer J., W. Schrettl and P.J.H. Schröder (2007), Johnson (2002), Myatt and Wallace (2008)). The basic argument runs as follows: Software of quality $V$ can be developed at cost $c_i$ by user $i$ who enjoys utility $V_i = \alpha_i V$ in return. There are $\eta$ potential users who differ with respect to their costs of developing the software and/or with respect to their valuation of the software. Essentially, the user with the highest ratio $V_i/c_i$ will develop the software and share it with all other users. The approach can be illustrated for $\eta = 2$. For simplicity we assume that both agents have the same development costs $c$, but their utility from the software differs, and without loss of generality we assume $V_1 > V_2$. If $V_1 > c > V_2$ the outcome is trivial: User 1 develops the software and user 2 adopts it for free as by assumption it is OSS. If $V_1 > V_2 > c$ the resulting game is slightly more complicated since the equilibrium in the static game is not unique. This can be seen from the following payoff-matrix.

Each user has two strategies: to develop or to wait. If one user develops and the other waits then the latter gets the software for free. If both wait then the software will not be developed.

<table>
<thead>
<tr>
<th></th>
<th>User 2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Develop</td>
<td>Wait</td>
</tr>
<tr>
<td>User 1</td>
<td>$V_1-c$, $V_2-c$</td>
<td>$V_1-c$, $V_2$</td>
</tr>
<tr>
<td></td>
<td>$V_1$, $V_2-c$</td>
<td>0,0</td>
</tr>
</tbody>
</table>

Clearly, for $V_1 > c > V_2$ in the unique equilibrium user 1 develops and user 2 waits, if $V_1 > V_2 > c$ then it is also an equilibrium that user 2 develops and user 1 waits. In addition, there exists also an equilibrium in mixed strategies: Denote the probability
that player 1 develops as $\omega_1$ and the corresponding probability for player 2 as $\omega_2$. The indifference conditions for the two players are

$$V_1 - c = \omega_2 V_1,$$

$$V_2 - c = \omega_1 V_2,$$  \hspace{1cm} (1), (2)

hence, in equilibrium

$$\omega_1 = \frac{V_2 - c}{V_2},$$  \hspace{1cm} (3)

$$\omega_2 = \frac{V_1 - c}{V_1}. $$  \hspace{1cm} (4)

However, if we move from a one shot game to a finite dynamic waiting game there exists a unique subgame-perfect equilibrium in which user 1 would develop immediately and user 2 would adopt afterwards (see Bitzer et al. 2007).

In an earlier version of this approach Johnson (2002) assumes that the type of a potential user is his private information. Denote the probability that a player is of type 1 as $\gamma$. The indifference conditions which have to be satisfied if a player randomizes between pure strategies are

$$V_1 - c = [\gamma \omega_1 + (1 - \gamma) \omega_2] V_1,$$

$$V_2 - c = [\gamma \omega_1 + (1 - \gamma) \omega_2] V_2.$$  \hspace{1cm} (5), (6)

Clearly, it is impossible that both conditions hold simultaneously. In particular, if (5) is satisfied the left hand side of (6) must be smaller that the r.h.s, and consequently, $\omega_2 = 0$. In equilibrium we get therefore

$$\omega_1 = \min \left\{ 1, \frac{V_1 - c}{\gamma V_1} \right\},$$  \hspace{1cm} (7)

$$\omega_2 = 0.$$  \hspace{1cm} (8)
It is straightforward to generalize the above models for an arbitrary number of players $n > 2$ (see Johnson 2002).

While this approach provides an explanation for the emergence of an OSS-project it begs the question why the software would not be developed by a commercial firm, or why user 1 is willing to give away the software for free. A commercial software firm could sell the software at a price $V_2$ to both users or at price $c$ only to user 1 and break at least even. But even if the commercial firm would not develop the software because $c > 2V_2$ it is unclear why user 1 should not sell the software at price $V_2$ to user 2.

One answer is that participating as a developer in an OSS-project yields additional benefits in terms of reputation in the community and/or future income because of signaling the quality as a software programmer. As far as the above models have only one user or one type of user developing the software even in the case with more than two potential users this argument is not entirely convincing in the current framework. We shall focus instead on two aspects of OSS which have been neglected in these models.

(i) Software is typically a network good with a positive externality, hence the utility of each user is increasing in the total number of users.

(ii) An often mentioned advantage of OSS is that its development is carried out by many programmers which is supposed to increase the efficiency of the development process and/or the quality of the software.

In the following section we take up both aspects in turn.
4. Positive network externalities in the use of software

Software is frequently considered as a typical example of a network good with the property that its utility for at least some users is increasing in the total number of users (see e.g. Shy 2001). Such a positive network externality may be direct – e.g. users can exchange files etc. – or indirect – e.g. the more users there are the greater is the incentive to improve the software or to develop complementary products. With OSS a positive network externality is supposed to exist also in the development phase as potential users have access to the source code and can make modifications and improvements which are available for all users. We shall analyze in more detail the production of software in the next section. Currently we show how the adoption games of the previous section are modified if positive network externalities are taken into account.

Suppose as above that there are $\eta$ potential users of some software. The utility of user $i$ if he gets the software equals

$$U_i(q) = V_i + \nu_i(q - 1),$$

where $q$ denotes the total number of users of the software and $\nu$ the marginal utility of the number of users. As before we assume that the costs of developing the software are $c$. Let us start with the case in which $\eta = 2$, $V_1 \geq V_2$ and $\nu_1 \geq \nu_2$ with at least one strict inequality. Considering an OSS-project first each player has two strategies: develop at cost $c$ or wait. If he waits and the other user develops the software he gets it for free, otherwise he gets nothing. The payoff-matrix looks as follows:
Now if $V_1 > c$ we have essentially the same situation as described in the previous section: As long as $V_2 + v_2 < c$ the unique equilibrium has user 1 developing the software and user 2 waiting, otherwise two equilibria in pure strategies and a mixed strategies equilibrium exist.

To see the potential importance of the positive network externality assume next $V_1 + v_1 > c > V_1$ and $V_2 + v_2 < c/2$. In this case the equilibrium is unique with user 1 developing the software and user 2 getting it for free.

Under these assumptions a commercial software firm would never develop the software as it would incur losses. It could sell at price $V_1$ to user 1 only, but by assumption this is less than the costs of development. Alternatively it could sell at price $V_2 + v_2$ to both users, but again it would not cover the development costs. The argument can easily be extended to n players. A commercial firm with n potential users would not develop software if $V_k + (k-1) v_k < c/k$ for $2 \leq k \leq n$. We summarize our findings in the following

**Proposition 1:** Suppose software generates positive network externalities for its users, and there is at least one user whose utility including the network effect exceeds the cost of developing the software. Then there exist values for $V_i$, $c$ and $v$ such that OSS would be developed whereas CS would not be profitable.
Now while the above model shows that an OSS-project might work under conditions which would make CS unprofitable it is still an open question why user 1 gives away the software for free. There are several answers to this.

(i) Remaining within the basic model one could augment it by adding a bargaining game between the two potential users. Depending on how this is specified giving away the software for free can easily be a Nash equilibrium. Consider, for example, a version of the ultimatum game in which user 2 makes an offer, and user 1 either accepts or rejects. In any case, in comparison to the version without network externality player 2 has more bargaining power.

(ii) Not all OSS-projects use the most restrictive license GNU which requires each participant to make the software he develops and the source code freely available to everybody. Take for example Berkeley Software Distribution (BSD): the source code is freely available, anyone can modify the program and distribute it for a fee, the original source must be acknowledged, but the source code may be kept. In our simple model this may well be a rational strategy against a commercial software firm: The two players may reach an agreement beforehand that player 1 shares the software for a fee. In fact this would weaken commercial software firms even more, because there would not be a profitable entry into the market even if \( V_2 + v_2 > c/2 \). If the two potential users agree on a fee of \( c/2 \) no profitable entry of the commercial developer is possible.

(iii) Finally, it may well be that the network externality generated by OSS is greater than for closed software. This is the case if user 2 would make
improvements which require the source code and which also benefit user
1. In fact, it is often argued that a main advantage of OSS over commercial
software is that the quality of software is increasing in the number of
programmers who participate in its development, however small each
individual contribution may be. We take up this point in the next section.

In any case, taking the network externality of software into account makes the public
good approach to OSS-projects more convincing, in particular with respect to the
possible superiority of OSS over commercial software if there is (potential)
competition between the two. It remains a shortcoming, however, that in equilibrium
one potential user develops the software alone (in a mixed strategies equilibrium more
than one player may do so, but they don’t co-operate). On the other hand, a main
feature of OSS-projects is the co-operation of many programmers. In order to address
this aspect we turn next to the production side of OSS.
5. Production of software

So far we have assumed that the quality of the software is given (though its value may be different for different users) and the costs of producing it are independent of the organization of production. It is often argued, however, that software production becomes more efficient if the number of programmers participating in its development is increased, at least within certain limits. The reason could be that for certain problems more alternatives are tried, more ideas are pursued and different experiences can be utilized. We call this effect the “economies of collaboration”. On the other hand, of course, if the number of participants gets too large duplications of efforts and coordination problems reduce efficiency, i.e. the economies of collaboration are offset by diseconomies of scale. Increased efficiency can be seen in two ways. Assuming that the time of developers is the only factor of production either total labor input is reduced for a given quality of software and/or the quality of the software is higher for a given total labor input. In this paper we use the first approach, keeping in line with the simple model of the previous sections.

5.1 Exogenous quality of software

5.1.2 Two potential users

We use the following notation: The total number of hours worked is denoted as \( x \), the number of programmers participating as \( n \). The number of hours required to produce the software is given by the strictly convex function \( x(n) \), with \( x'(1) < 0 < x'' \), and \( n^* \) is the number of participants that minimizes \( x \). The value of one hour of programming time is denoted as \( w \). As far as commercial software firms are concerned it is assumed that hiring a programmer requires fixed costs \( h \). This can be thought of as the costs of
advertising, screening, providing a working place etc. It is assumed that for an OSS-project such costs are substantially smaller, for the sake of simplicity we assume them to be equal to zero. The total costs of producing the software are

\[ c^c = wx(n^c) + hn^c \]  

(10)

for the commercial firm, and

\[ c^O = wx(n^O) \]  

(11)

for OSS.

Depending on the number of participants in the OSS-project the costs of developing the software may be greater or smaller than for a commercial firm. However, since OSS need not make a profit it may be provided even if CS would not be developed. We illustrate this possibility by modifying the above game with two potential users in the following way: The cost function is given by

\[ c^c = w\left[\bar{x} - a(n-1) + (n-1)^2\right] + hn, \]  

(12)

and the minimum is reached at \( n^c \) satisfying

\[ n^c = \frac{a}{2} + 1 - \frac{h}{2w}. \]  

(13)

The number of employees for the CS is increasing in the parameter \( a \) which captures the marginal cost reduction at \( n = 1 \), and in the ratio of hiring costs to the wage rate. For our example let \( a = 2 \) and \( h = 2w \), consequently

\[ n^c = 1, \]  

(14)

and

\[ n^O = 2. \]  

(15)
The utility of user \( i \) is given by (9). For the sake of simplicity let the marginal network effect \( \nu \) be the same for both users. Now assume

\[
 e^c = w\bar{x} + h > w\bar{x} > 2\nu + V_1 + V_2 > w[\bar{x} - 1]. \tag{16}
\]

Condition (16) implies that CS could never make a profit. However, even if a potential user would save on the hiring costs and could enjoy the consumption network externality of the software he would not produce it as OSS. Only the additional network externality of production would make developing the software as OSS worthwhile.

The question is whether the OSS-project would ever get started. Note first that it is a Nash-equilibrium that neither user contributes to the development of the OSS. On the other hand there is a continuum of pure strategies equilibria with both users participating in the OSS-project. The upper limit of programming time user \( i \) is prepared to contribute equals

\[
 \bar{x}_i = \frac{\nu + V_i}{w}, \tag{17}
\]

i.e. the value of his labor input cannot exceed his evaluation of the OSS. The lower limit for the contribution of user \( i \) which is compatible with the development of OSS, denoted as \( \underline{x}_i \), is defined as follows.

\[
 \underline{x}_i = [\bar{x} - 1] - \bar{x}_i. \tag{17}
\]

Now any combination of \( x_i \) and \( x_j \) satisfying

\[
 \underline{x}_i \leq x_i \leq \bar{x}_i \text{ and } x_i + x_j = \bar{x} - 1 \tag{18}
\]
is a Nash-equilibrium. Clearly, if decisions have to be taken simultaneously there is a formidable coordination problem. In reality, however, decisions have rarely to be taken without any information about actions and intentions of others. In fact, it is the very strength of the internet to allow for fast and efficient dissemination of information. In addition, all OSS-projects require a minimum of structuring of tasks. In the framework of our simple model assuming sequential moves is already sufficient for getting the OSS-project going. Suppose user 1 moves first and transfers the outcome of his effort $x_1$ to user 2. Clearly, as long as $x_1$ satisfies (18) it is a best response of user 2 to complete the OSS.

Now a rational player would try to exploit his “first mover advantage” by choosing the smallest $x_1$ which induces the second player to complete the software. It is commonly observed, however, that those who initiate an OSS-project provide a particularly large input for the development of the software. One explanation that follows immediately from our simple model is that they are those who enjoy the greatest benefit, hence they are willing to make the largest contribution. A second explanation may invoke concepts of “fairness” or of “gift exchange” which imply that a player does not fully exploit his first mover advantage. In our framework we do not account for such motives and stick to a standard game with a given sequence of moves.

5.1.2 Endogenous Number of Contributors

The above example was chosen such that a commercial software firm would employ too few programmers whereas the number of potential users is such that total costs of the OSS-project are minimized if they all take part in its development. In reality this is very unlikely to happen. In fact, one of the arguments against the efficiency of OSS-
projects was that there are substantial diseconomies of scale and that the development costs become very large if too many programmers participate. This is reflected in the cost function (11) above. Using our simple numerical example it is easy to see that the costs of the OSS increase very quickly if there are more than 2 potential users who participate actively in the development. For \( n^O = 3 \) we already get \( x^O(3) = \bar{x} \), though the OSS-project is still cheaper than the corresponding CS as there are no hiring cost. For \( n^O = 4 \), however, the joint development of OSS very quickly becomes very costly, also in comparison to the CS. In fact, two questions need to be answered.

a) Under which conditions does a Nash-equilibrium exist such that OSS will be developed?

b) Given such an equilibrium, do there exist conditions such that CS would not be viable?

The answer to the first question is found in a straightforward extension of (18) to the case of \( n \) potential users which is given in the following proposition.

**Proposition 2:** A Nash-equilibrium in which the OSS is developed exists in the sequential software developing game if and only if there exists some \( k^* \) such that

\[
x(k^*) \leq \sum_{i=1}^{k^*} \bar{x}_i.
\]  

(19)

The Nash-equilibrium is not unique. As mentioned earlier it is also a Nash-equilibrium that no OSS is developed at all. On the other hand any combination of labor inputs of the first \( k^* \) players that adds up to \( x(k^*) \) such that no individual input exceeds \( \bar{x}_i \) is also a Nash equilibrium. Finally, \( k^* \) need not be unique. In this paper we shall not dwell on the problem of the selection of an equilibrium and give just one
example how an OSS-development game could be designed that ensures an efficient development process. For this purpose we model the development process as a sequential game. Suppose players move in the order of their valuation of the software, which is assumed to be public knowledge. Knowing that the addition of one or more additional programmers would increase the total labor requirement substantially and may even endanger the success of the project the user whose turn is close at the stage at which the OSS should be completed if costs are to be minimized has indeed an incentive to do so, provided the programmers at earlier stages take this into account and choose their contribution accordingly. We use the following simple game to illustrate possible scenarios. Suppose there are $\eta$ potential users who can be ordered according to their valuation of the software such that $V_1 > V_2 > \ldots > V_{\eta-1} > V_\eta$. They play the following game: In stage 1 user 1 devotes programming time $x_1$ to developing the software. If $x_1 > 0$ he passes on the result of his effort to player 2 who chooses $x_2$ and passes on the result to player 3 who chooses $x_3$ etc. The game ends if either the software is finished and becomes available to all potential users, or if no player is willing to carry on the project. In this case no OSS becomes available. Now suppose it is the turn of player $k$. He has three possibilities:

1. He can choose to contribute nothing and pass on what he got without any modifications to player $k+1$.

2. He can finish the software and the game ends.

3. He can contribute to the development of the software and then pass it on to player $k+1$. 

Action 1 makes sense if he believes that the software will be completed also without his contribution and he can free ride. It also makes sense if he believes the software will never be completed even if he contributes his maximum effort $\bar{x}_k$. The second action makes sense if his contribution for finishing the software is smaller than $\bar{x}_k$ and he believes the next player would not complete the software. The third action makes sense if he believes that the software would not be completed without his contribution. Now consider the choice between action 2 (finish the software) and action 3 (contribute and pass on). Denote the necessary total labor input if there are $k$ active participants as $x(k)$. In order to finish the software he would have to contribute $x_k$ satisfying

$$x_k = x(k) - \sum_{i=1}^{k-1} x_i$$  \hspace{1cm} (20)$$

If he decides to contribute to the project in order to induce the next player to finish it he has to invest

$$x_k = x(k+1) - \sum_{i=1}^{k-1} x_i - \bar{x}_{k-1}$$  \hspace{1cm} (21)$$

Subtracting (20) from (21) reveals that player $k$ will prefer to complete the software himself if

$$x(k+1) - x(k) \geq \bar{x}_{k+1}$$  \hspace{1cm} (22)$$

Suppose $k > n^*$, i.e. $x$ is already increasing in $k$. Since $x(k)$ is strictly convex and $\bar{x}_{k+1} > \bar{x}_{k+2}$ it would not make sense for player $k$ not to complete the software if (22) holds, because the gap between the labor input required to finish the OSS-project successfully and the maximum input any potential user $j > k + l$ is willing to
contribute would be increasing. Hence, the first player for whom (22) holds would end the game. Consequently, no more than the first $k$ users would become active in an equilibrium in which the software is produced. Player $k-1$, knowing this, would make a contribution such that (20) holds for $x_k$. Hence, he would choose $x_{k-1}$ satisfying

$$x_{k-1} = x(k) - \sum_{j=1}^{k-2} x_j - \bar{x}_k$$  \hfill (23)

But player $k-2$ would employ exactly the same reasoning, thus

$$x_{k-2} = x(k) - \sum_{h=1}^{k-3} x_h - \bar{x}_{k-1} - \bar{x}_k$$  \hfill (24)

Working this way backward to player 1 we get

$$x_1 = x(k) - \sum_{i=2}^{k} \bar{x}_i$$  \hfill (25)

A Nash-equilibrium can therefore be described as follows: The agent with the highest evaluation of software starts the game by investing $x_1^*$ as defined by (25). The following players contribute sequentially $\bar{x}_i$, $i = 2, \ldots, k^*$ such that player $k^*$ completes the software. Player $k^*$ is defined by the following inequalities:

$$x(k^* - 1) - x(k^*) < \bar{x}_{k^*}$$  \hfill (26)

$$x(k^*) - x(k^* + 1) > \bar{x}_{k^* + 1}$$  \hfill (27)

It is easy to see that the equilibrium in which player 1 starts with $x_1^*$ and each of the following players supplies his reservation labor input is the only sub-game perfect equilibrium.
The question we want to address in this section concerns the possibility that an OSS-project can succeed where CS would fail to make a profit. This is in fact possible, but the reverse may also happen. To see this consider first the condition for CS to be profitable which is given in the following proposition.

**Proposition 3:** Suppose there are \( \eta \) potential users who can be ordered according to their valuation of the software such that \( V_1 > V_2 > \ldots > V_{\eta-1} > V_\eta \). A necessary and sufficient condition for profitable CS is the existence of user \( m \) satisfying

\[
m[(m - 1)v + V_m] > c^e. \tag{28}
\]

Proof: The expression in square brackets equal the reservation price of user \( m \). At this price all users \( i < m \) would also buy as they attach a higher value to the software than user \( m \). Consequently, the left hand side of 27 is the highest return possible for CS if it is sold to \( m \) users. A profit requires that this return is greater than the costs of production.

As we demonstrate below by means of a simple example, conditions (27) and (28) neither imply nor exclude each other. Consequently, we get the following proposition.

**Proposition 4:** Suppose there are \( \eta \) potential users who can be ordered according to their valuation of the software such that \( V_1 > V_2 > \ldots > V_{\eta-1} > V_\eta \). The costs of developing the software are given by (12). Then it is possible, that only OSS is viable, only CS is viable, both are viable or neither is, depending on which of the conditions (29) and (28) are satisfied.

Proof: see the following example.
Example 1: In order to illustrate the various possibilities consider the following numerical example. There are four potential users. Using the cost function (12) we assume the following parameter values: $\bar{x} = 20$, $a = 2$, $h = 2$, $w = 1$. Depending on the number of active developers we get $x(1) = 20$, $x(2) = 18$, $x(3) = 20$, $x(4) = 23$, hence $x(2) - x(1) = -2$, $x(3) - x(2) = 2$, $x(4) - x(3) = 3$. It is straightforward to find values for $x_i$ such that 1, 2, 3 or 4 potential users become active or none at all. If $x_i \geq \bar{x}$ and $x_j = 0$ for $j \geq 2$ then we are essentially back in the world of section 3 with player 1 developing the software and giving it away for free to all other potential users. For $x_1 = 20, x_2 = 8$ and $x_3, x_4 < 1$ the software will be developed by the first two users who invest 10 and 8 respectively. Finally, if the reservation inputs are 6, 5, 4 and 3 then total labor input of the players is too small to create the OSS.

Consider next CS. We get $c^e = 22$. Note that under our assumptions the reservation price of user $i$ is equal to $\bar{x}_i$. Note that for all cases described above CS is not profitable. Now suppose next $x_i = 5.6$ for $i = 1,2,3,4$. In this case OSS would not be viable because the sum of all contributions is $22.7 < 23$, but CS would be profitable.

The example is instructive because it shows that OSS requires for its success a relatively small number of highly motivated users who can exploit the economies of collaboration before the diseconomies of scale become too effective. CS, on the other hand, is more likely to be viable if the users are not too heterogeneous and economies of collaboration are not very strong.

At this point the question arises how plausible it is that a fairly small number of potential users develops software for free when it is rather complex and requires a large number of working hours. Even if contributors to OSS derive additional benefits from their participation and value their opportunity costs of 1 hour of programming by
less than the market wage \( w \) it may still require a very large number of active
participants to supply the required labor input, and this number may exceed the
efficient number by far. One answer to this question is modularization. Very often
software can be divided into several parts which may be developed independently
from each other. The task of the initiator(s) of an OSS-project is then, of course to
structure the process appropriately. But if it is possible to divide the development of
OSS into smaller tasks which can be assigned to smaller groups of volunteers the
economies of collaboration may still be strong enough to make the production of OSS
more or at least as efficient as that of CS. A detailed analysis of this aspect, however,
is beyond the scope of this paper and has to be left to future research.

5.2 Choice of Quality

It is sometimes claimed that OSS has better quality than similar OS. Leaving aside the
non-trivial issue how to measure the quality of software we assume that it is possible
to rank software according to quality. To fix ideas assume that it is possible to
develop software for a particular purpose in two qualities denoted as \( V_H \) and \( V_L \), with
\( V_H > V_L \). The cost of developing the software are denoted as \( c_H \) and \( c_L \) respectively,
with \( c_H > c_L \). There are \( \eta \) potential users of the software who differ with respect to
their valuation of quality. The marginal utility of quality for consumer \( i \) is denoted as
\( \alpha_i \), and for the sake of simplicity we assume that \( \alpha \) is uniformly distributed over the
interval \([0,1]\). Furthermore, to keep things simple, we assume in this section that there
are no consumption externalities.

A social planner maximizing total net utility would choose the higher quality \( V_H \) if
and only if
\[ \eta(V_H - V_L)/2 \geq c_H - c_L. \]  \hspace{1cm} (29)

The left hand side of (29) is the gross welfare difference between the two software qualities which equals the average difference times the total number of users, the right hand side is the difference in the costs of production.

A commercial software developer would produce the quality which yields the higher profit. Abstracting from consumption externalities the marginal consumer of the software who is indifferent between buying and not buying at price \( p \) is defined by the condition

\[ \alpha(p, V_i) = p/V_i. \]  \hspace{1cm} (30)

All consumers with \( \alpha_j \geq \alpha(p, V_i) \) would buy software of quality \( V_i \) at a price \( p \).

Consequently, the commercial software firm would maximize

\[ \max \pi(p, V_i) = \eta[I - p/V_i]p - c_i, \]  \hspace{1cm} (31)

which implies the commercial software price \( p^*(V_i) \)

\[ p^*(V_i) = V_i/2 \]  \hspace{1cm} (32)

The corresponding profit equals

\[ \pi(V_i) = \eta V_i/4 - c_i. \]  \hspace{1cm} (33)

Consequently, a monopolist would produce the high quality OS if and only if

\[ \frac{\eta}{4} (V_H - V_L) \geq c_H - c_L. \]  \hspace{1cm} (34)

Comparing (34) and (29) shows immediately the following result:
**Proposition 5:** If there are no positive consumption externalities a monopolist will never produce CS with socially inefficient high quality.

Conversely, he is quite likely to produce socially inefficient low quality CS for two reasons: In contrast to the social optimum he does not provide all potential users with the software, and he can only capture part of the total consumer surplus.

We turn now to the question which quality would be chosen for OSS. Note first that if the software is developed by a single user – the case assumed in the public goods models analyzed by Bitzer et al. (2007) and Johnson (2002) – the OSS would not have higher quality than CS for \( \eta \geq 4 \): A single developer of OSS would choose the high quality if and only if \( V_H - c_H \geq V_L - c_L > 0 \), which implies (34) for \( \eta \geq 4 \). The more interesting case, however, arises when more than one potential user has to contribute to the development of the OSS. As before we assume that the user with the highest \( \alpha \) starts the game, but now he has not only to decide how much effort to invest, but also which project to start. He will choose the high quality software if and only if

\[
V_H - wx_1^H \geq V_L - wx_1^L, \tag{35}
\]

where \( x_1^S \) denotes the smallest labor input required in order to ensure that OSS of quality \( S \) will be produced in equilibrium, \( S = H, L \). As shown above, \( x_1^S \) satisfies

\[
x_1^S = x(k^S) - \sum_{j=2}^{k^S} \pi_j^S, \tag{36}
\]

where \( k^S \) is defined as in (26) for software \( S = H, L \). Now recall the definition of \( \pi_j^S \) given by (17) for \( \nu \) equal to zero:
\[ x^S_j = \frac{\alpha_j V^S_j}{w}, S = H, L. \tag{17'} \]

Observe that the costs of producing OSS of quality \( S \) are equal to

\[ c^O_S = w(x^{k^S}). \tag{37} \]

Assuming that there is a finite number of users we need a distribution of \( \alpha \in [0,1] \)
which corresponds to the uniform distribution for a discrete random variable. We get

\[ \alpha_i = \frac{\eta - i}{\eta - 1} \tag{38} \]

Note that the average \( \alpha \) for all users with \( \alpha_i \geq \alpha_k \), denoted as \( E[\alpha | \alpha \geq \alpha_k] \) is given by

\[ E[\alpha | \alpha \geq \alpha_k] = \frac{2\eta - 1 - k}{2(\eta - 1)}. \tag{39} \]

Substituting (36) – (39) and (17') in (35) and rearranging terms yields the following condition for developing high quality OSS

\[ \frac{4(\eta - 1) - k^H}{2(\eta - 1)} (k^H - 1)V^H - \frac{4(\eta - 1) - k^L}{2(\eta - 1)} (k^L - 1)V^L \geq c^O_H - c^O_L \tag{40} \]

A comparison between (34) and (40) shows that even with a uniform distribution of the marginal utility of software quality it is not possible to identify any general relations between the two conditions as they neither imply each other nor do they rule each other out. In view of Proposition 4 this is no surprise as it was not possible to rule out that OSS would not be viable where CS is. What can be said is that OSS is more likely to be of high quality the larger the number of active developers is, both in absolute terms and relative to the number of developers in the corresponding low quality project, and the smaller the difference between the development costs is.
Without specification of the cost functions it is not possible to determine $k^S$ and $c^O_S$.

Consequently, we get a result analogous to Proposition 4:

**Proposition 6:** Suppose there are $\eta$ potential users who can be ordered according to their marginal valuation of software such that $\alpha_1 > \alpha_2 > \ldots > \alpha_{\eta-1} > \alpha_{\eta}$. The software can be provided in two qualities, $V_H$ and $V_L$. The costs of developing the software are given by (12) and (42). Then it is possible, that only OSS would have quality $V_H$, only CS would have quality $V_L$, both would have high quality or both would have low quality, depending on which of the conditions (34) and (39) are satisfied.

Proof: See the following example.

**Example 2:** Using cost function (12), assume

$$x^S(n) = \bar{x}_S - a(n-1) + b(n-1)^2, \quad S = H, L.$$  \hspace{1cm} (41)

Let $\eta = 11$, with $\alpha_1 = 1$, $\alpha_2 = 9/10$, $\ldots$, $\alpha_{10} = 1/10$, $\alpha_{11} = 0$. Furthermore, $V_H = 14$, $V_L = 10$, $\bar{x}_H = 38$, $\bar{x}_L = 25$, $a = 2$, $b = 1$, $w = 1$, $h = 2$.

As has been shown above, a commercial firm would hire one developer, hence

$c^H_{\eta} = 40, c^L_{\eta} = 27$. For the low quality software it would either charge price 5 and sell 6 copies or charge price 6 and sell 5 copies. In either case, it would make a profit of 3.

For high quality it would charge a price of 7 and sell 6 copies yielding a profit of 2.

In an open source project there would be 4 contributors for each quality. To see this note that $x(4) - x(3) = 3$ and $x(5) - x(4) = 10$ for both qualities, and the maximum input of user 4 is 7 for low quality and 9.8 for high quality. User 5 is ready to contribute 6 units of labor for low quality software and 8.4 for high quality.
Consequently, we get $c_H^O = 40, c_L^O = 27$. In an OSS equilibrium with low quality users 2 – 4 would supply together 24 units of labor input, leaving 3 units for user 1, who would gain a net utility of 7. In the high quality equilibrium players 2 – 4 would together contribute 33.6 of the required 40 units of labor input, requiring a labor input of player 1 of 6.4 and leaving him with net utility 7.6, hence OSS would be of high quality whereas CS would be low quality. We leave it to the reader to modify the example in such a way that the other possibilities stated in proposition 6 arise.

In any case, we have shown the possibility that an OSS-project aims for higher quality than a CS-developer. A more detailed analysis would have to be more precise with respect to the definition of “quality”, as OSS and CS are likely to focus on different properties and characteristic of software.

We turn next to the question whether OSS may be able to enter a market from which a CS-developer would stay out.
6. Market entry

So far we have compared OSS and CS under the assumption that either one or the other is developed. In reality, however, quite often OSS and CS compete with each other. In particular, it is often observed that some CS is holding a (temporary) monopoly which is challenged by OSS, Linux being probably the best known example. It is well known that it is very hard to enter a market for a good with network externalities once a particular product is firmly established, even if a better product is available. The reason is a lock-in effect as described first by B. Arthur (1989), i.e. a network good needs a critical mass of users – the installed base – in order to be successful. In addition, especially less sophisticated users may face considerable switching cost. Using insights from previous sections we show that OSS may successfully enter a market under conditions which would not allow for entry of CS. Suppose that currently CS has \( \eta \) users. Its quality equals \( V_L \), and the utility to user \( i \) equals \( \alpha_i V_L + (\eta - 1) \nu \), \( i = 1, ..., \eta \), \( \alpha_i > \alpha_{i+1} \). A switch to alternative software would induce switching costs \( \sigma_i \), \( i = 1, ..., \eta \), \( \sigma_i < \sigma_{i+1} \). Suppose next an alternative CS with quality \( V_H > V_L \) could be developed at cost \( c^e \). A necessary condition for successful entry is the existence of an equilibrium such that \( k \) users switch to the new software at a price \( k \) which covers the development costs of the new software. Formally, we get the following condition.

\[
\alpha_k (V_H - V_L) - (\eta - 2k + 1) \nu - \sigma_k > \frac{c^e}{k}
\]  

(42)

The interpretation of condition (41) is as follows: at software price \( p \) user \( k \) is indifferent between staying with \( V_L \) and switching to \( V_H \) if

\[
\alpha_k V_H + (k - l) \nu - \sigma_k - p = \alpha_k V_L + (\eta - k) \nu.
\]  

(43)
Solving (43) for \( p \) yields the left hand side of (42). In order to make market entry at this price profitable \( kp \geq c^e \) is required.

It is noteworthy that even if (42) is satisfied it is still a Nash-equilibrium that no user switches to the new software. In reality this non-uniqueness of equilibrium and the uncertainty about which equilibrium will be realized may well stop commercial software developers from attempting market entry even if it could be profitable. The case we are interested in, however, is one in which (42) is not satisfied and yet OSS may successfully enter the market. We show next that under our assumptions this is indeed possible.

**Proposition 7:** Consider a software market with \( \eta \) users with utility functions (9) who are currently all using software of quality \( V_L \). Suppose software of quality \( V_H \) can be produced according to the cost function (12). Then there exist parameter values for the distribution of \( \alpha_i, \sigma_i \), such that (42) does not hold, but OSS can enter the market in equilibrium.

The proof is given by the following

**Example 3:** We are using the same specification of the cost function as in example 1.

Let there be 5 users with \( \alpha_1 = 1, \alpha_2 = 1/2, \alpha_3 = 1/4, \alpha_4 = 1/8, \alpha_5 = 1/16 \), and switching costs \( \sigma_1 = 0, \sigma_2 = 1, \sigma_3 = 5, \sigma_4 = 6, \sigma_5 = 8 \). Recall that the cost of production are \( c^e = 22, c^o(1) = 20, c^o(2) = 18 \). Let \( V_H - V_L = 20 \).

Note first that there is no profitable market entry for CS. The reservation prices of the users for switching to the better software are \( p_1 = 16, p_2 = 6, p_3 = -2 \) etc., thus a commercial software developer could never recover his costs.
Turning to OSS the reservation inputs of the first two users are $\bar{x}_1 = 16$, $\bar{x}_2 = 6$.

Clearly, it is an equilibrium that player one contributes developing time $x_1 = 12$, player 2 contributes $x_2 = 6$, and the other three users stay with the incumbent software.

As in previous examples it is straightforward to change this example in various ways in order to generate different outcomes, e.g. OSS is not feasible whereas CS is, or all users switching to the better software. What is intuitively clear from this example is that OSS requires a dedicated group of potential users who attach high utility to the quality of software. The number of active developers must not be too large in order to avoid that diseconomies of scale dominate economies of collaboration excessively, but it must be sufficiently large to create an installed base.
7. Concluding Remarks

There is no general answer to the question whether OSS or CS is superior with respect to quality, viability or development costs. OSS is favored by heterogeneity of users and is most likely to succeed if the “right” number of contributors can be motivated. The “right number” means that there is a balance between the economies of collaboration and the diseconomies of scale. In addition, if market entry is attempted, the creation of an installed base is vital.

Of course, our model is in many respects far too simple to provide final answers. In particular, in future research the following aspects need to be addressed.

- A detailed analysis of the development of software is required. In particular, the possibilities of modularization of software are essential for assessing the viability of OSS projects.

- Quality of software has many dimensions, and an important question concerns the choice of features and properties which receive special attention in OSS and CS respectively.

- The organization of the development of OSS requires a closer analysis. Examples are the self-selection of contributors, the choice of solutions which are incorporated in the software, and quality control.

- An important aspect is the co-existence of OSS and CS, who may compete, but often are also complementary.

Though much remains to be done we are confident that the basic insights of the present paper remain valid.
References


Myatt, D.P. and C. Wallace (2008), An evolutionary analysis of the volunteer’s dilemma, Games and Economic Behavior 62, 67-76.


