Open vs. Closed Source Software: Two Co-Existing, Copyright-Based Kinds of Organizations

Sebastian v. Engelhardt *

July 4, 2007

WORK IN PROGRESS, COMMENTS ARE WELCOME. PLEASE DO NOT QUOTE.

Abstract

The software industry faces the co-existence of 'closed source software' (CSS)—or: 'proprietary'—and 'open source software' (OSS). The latter is developed by communities incl. hobbyists as well as companies, and the source code, the human-readable recipe of a software, is 'open' (disclosed). OSS and CSS are different kinds of institutional arrangements, distinguished by their use of copyright law, codified in the software licenses. These licenses therefore lead to different allocation of intellectual property rights and different modes of organization. The different institutional arrangements represent different strategies in use of the resource software, source code respectively.

The paper outlines an analytical framework in order to analyze the co-existence of CSS and OSS. The economic characteristics of software and transaction costs explain, why CSS and OSS co-exist, as they are two different solutions for the same internalization problem. OSS vs. CSS is about the efficient use of a non- and anti-scarce resource, because of positive external effects. Some of this effects can not be internalized by market based allocation of property rights. CSS, as based upon exclusive use of source code, is limited in its scope of cooperation (number of members), while OSS is a non-specific cooperation contract offered to anybody and therefore has to minimize exclusive rights down to the level of passively controlled projects.

*Friedrich-Schiller-University Jena, Email: Sebastian.Engelhardt@wiwi.uni-jena.de

Contents

1 Introduction

> "If the main allocative function of property rights is the internalization of beneficial and harmful effects, then the emergence of property rights can be understood best by their association with the emergence of new or different beneficial and harmful effects."
>
> *H. Demsetz, Towards a Theory of Property Rights, p 350*

The software industry is characterized by the co-existence of two copyright-based types of production modes: Beside proprietary software there exists also so called open source software. Open source software (OSS) is developed by communities (incl. hobbyists as well as companies) and the source code—the human-readable recipe of a software program—is 'open' (disclosed), which means that everybody has access to the software and its source code and the right to read, modify, improve, redistribute and use it. Firms like IBM, HP, or Sun Microsystems are involved in the OSS community as well as OSS distributors[1] like RedHad or Novell's SUSE. One can interpret the production modes of CSS vs. OSS as being different kinds of "institutional arrangements" (Davis & North 1971), and distinguish them by their *different use of copyright law*, codified in the software licenses. This leads to different allocations of intellectual property rights (IPRs) and different modes of organization. The institutional arrangements represent strategies in use of the resource software (source code respectively) and have specific assets and drawbacks regarding individual and firm level as well as social welfare.

The property rights theory mostly concentrates on negative external effects and the widely discussed tragedy of the commons—as well as the tragedy of the anti-commons (Heller 1998)— is a negative externality story, a scarce resource story. This focus seems to draw back to Demsetz' seminal article: Although he points out in the beginning, that it is about "internalization of external costs and benefits" (Demsetz 1967, p 349), he then focuses on negative externalities (Demsetz 1967, pp 350 ff.). But the issue discussed in this paper is about a non-scarce, to some extent even an anti-scarce resource. My argument is, that property rights regarding some kind of non-rival and anti-rival applications of software (or: of the source code) are not exclusively

[1] Open source software distributors sell complementary goods, thus they collect and optimize given open source software and offer further services like support and maintenance.

separable, and this leads to a control problem—a de facto dilution of exclusive ownership—if one wants to create (or: enable) and internalize these positive effects via market transactions. Therefore I show, that neither non- nor anti-rivalry in use as such, are reason for the dichotomy in the software industry, as a perfect market would lead to a welfare optimal allocation of non- and anti-rival applications. In a next step, optimal defined, copyright based licensee agreements are derived from this optimal allocation in order to show, that an optimal allocation of property rights is theoretically possible as well. Based on this, it is argued, that neither limits in market trading because of ex ante transaction costs (search and bargaining costs), nor incomplete knowledge can reason such a crucial internalization problem, that this could explain the co-existence of CSS and OSS. Hence, the argument is, that because of ex post transaction costs some of the (optimal defineable) property rights are not exclusively separable, which leads to a control problem, i.e. a de facto dilution of exclusive ownership. The principle of CSS and OSS are therefore interpreted as being two different kinds of solution for this, two different governance structures. The former maximizes control and exclusive ownership while the latter minimizes control and exclusive ownership.

This paper is part of a general analysis, that is based on the view, that the value of the economic resource software (just as any resource) is determined by its applicability, thus by the set of *possible strategies* in use, i.e. by the (expected) *payoff* of those strategies. The set of strategies itself is limited by

1. the technical-physical characteristics of the resource.
2. the institutional boundaries, e.g. (intellectual) property rights.
3. the traits and conducts of the other players (e.g. express of preferences in demand, skills they may offer, strategies they use, etc.).

Of course, there are interdependencies, e.g. the strategies in use of a resource are limited by property rights institutions (Libecap 1989) and by its technical-physical attributes, but the latter determine the kind of external effects to be internalized, and transaction costs influence the scope of internalization and property rights institutions (Langlois 2002, Libecap 2004); institutions itself are result of human action and interaction (Aoki 2007, 2001), etc. Keeping in mind that such interdependecies exist, the analysis consists of consecutive building blocks. Hence, *the paper at hand*—providing a first draft of

the analysis—*is structured as follows*: After an introduction to the analytical framework (section 2), the analysis starts with the characteristics of software (section 3.1) and explains why a source code is a non- and anti-scarce resource (section 3.2). In section 4 the role of IPRs—namely copyright—with respect to the non- and anti-scarce resource software (source code) is discussed, and—as already mentioned above—it will be argued, that control problems caused by ex post transaction costs inhibit optimal property right allocation, as some of them are not exclusively separable (section 4.3). In section 5 I turn to the two different production modes that have evolved and established in the software industry. Both principles are interpreted as being two different solutions, that try to deal with the problem of non exclusively separable property rights. Both have their assets and drawback and their limits. This gives reason to the fact, that OSS and CSS co-exits. With this background, the next step should be to take into account the traits and conducts of the other players. This yields a modeling of the two competing and co-existing, types of institutional arrangement, based on the individual OSS vs. CSS firm strategy decision. Section 6 provides a short outlook to that.

In general, focusing on institutional aspects of (open source) software is to some extent the complement to research dealing with the motivations of open source contributors: software is produced within institutional arrangements supporting different kinds of (innovative[2]) activities, supporting this or that kind of behavior based on different motives. A sound survey on motive-related as well as institutional aspects, theoretical and empirical research respectively, can be found in Rossi (2006). Gehring (2006) presents with his article on the "Institutionalization of Open Source Software" a brief introduction on institutional aspects, including IPRs, organizations and network standards as well as the code. Widely known is Weber (2004) with his analysis on the "Success of Open Source", documenting and examining the collaborative methods in context of developing OSS. One should also mention Wendel de Joode et al. (2003) in this context. (Of course, there is also a broad range of literature dealing with topics on managing OSS projects and/or governance problems and structures in OSS projects from a practi-

[2]Thus, a third important aspect related to this issue is research on open source and closed source/proprietary software as different innovation systems (institutional arrangements within innovations occur). See e.g. von Hippel & Von Krogh (2003), who point out, that OSS is a mixed innovation model, they call it "private-collective" innovation model.

tioner's perspective, mostly written from and for people involved in that area.) An organizational economist's view on Linux is provided by Franck & Jungwirth (2001), and Brand & Schmid (2005) discuss the cooperation in open source projects, based on a case study on the KDE project.[3] They distinguish between types of organization, types of coordination and instruments of coordination in order to describe how open source projects work. OSS communities are often interpreted as being networks and networks are often seen as a hybrid form between market and the firm, but Powell (1996) and Benkler (2002) argue that networks are a separate kind of organization, not a mixture. Especially the latter asserts that OSS is an example for commons-based peer production, which is a new, third mode of production in the digitally linked environment, and distinguishes this clearly from the property- and contract-based modes of firms and markets. Garzarelli (2003) also points out the uniqueness of OSS, arguing that its organizational characteristics are 'out of the ordinary' and can be explained by a combination of the organizational theory on clubs with the theory of professions. Thus, most of the literature concentrates on OSS and tends to express its uniqueness. An exception not dealing with OSS but focusing on institutional aspects is Engel (2002), who interprets network standards to be institutions—see also Wey (1999) about this—and argues that Microsoft's Windows is an institution organizing markets for application software: in order to get access to a network with a large installed base, software products need to be made compatible with only *one* standard, i.e. Windows. Some authors even interpret code to be a standard (Gehring, 2006, pp 67 ff, Lessig, 2006, 1999).

However, the paper at hand outlines an analytical framework to analyze the different institutional arrangements of the software industry in an integrated way, i.e. OSS *and* non-OSS. The aim is to examine the lasting co-existence of OSS and non-OSS with respect to the following research questions: *How can the co-existence be explained and what can be learned about the concepts of copyright-based ownership and active vs. passive control rights in this context?* Furthermore the analytical framework should be able to provides a basis for answering a second question: *Why and when do firms choose OSS or*

[3] KDE (K Desktop Environment) is an open source graphical user interface (GUI). Together with the GNOME desktop it is the likely most known desktop environment (and development platform) for Linux and Unix workstations (Webpage: www.kde.org).

2 The Analytical Framework

This section provides some definitions of the analytical framework used in this paper. Although it was developed in order to analyze the resource 'source code', the model can be used to analyze any kind of economic resource.

2.1 Modeling an Economic Resource

A resource can be broadly defined as a technically meaningful set of certain element, e.g. a source code can be broadly defined as a technically meaningful set of code lines. Therefore, in this model a given source code is described as a set X. As X can be split up into subsets, there is a set of all subsets $\mathcal{P}(X) = \{A \mid A \subseteq X\}$.

Let $y = f(Z, \cdot)$ denote that $Z \in \mathcal{P}(X)$ is used for an application y. The 'use' $f(Z, \cdot)$ is one of several possible forms of transformation of Z with or without the use of *other* code lines, e.g. $y = f(Z, W)$ would be an application of the combined code line sets Z and W, which can be rewritten as $y = f(V)$ with $V = \{\{Z \cup W\} \setminus U\} \neq \emptyset$. However, $y = f(Z)$ is possible as well.

Because of technical reasons, not for all $Z \in \mathcal{P}(X)$ exist applications, the trivial example is $Z = \emptyset \in \mathcal{P}(X)$. Additionally, there can be $Z \in \mathcal{P}(X)$ with multiple applications. This leads to the following:

Definition 2.1. The set of technically *not* meaningful subsets of X is

$$\mathcal{U}(X) = \{Z \in \mathcal{P}(X) \mid \nexists\, y = f(Z, \cdot)\}. \tag{1}$$

Definition 2.2. The set of technically meaningful subsets of X is

$$\mathcal{X}(X) = \{\mathcal{P}(X) \setminus \mathcal{U}(X)\} = \{Z \in \mathcal{P}(X) \mid \exists\, y = f(Z, \cdot)\}. \tag{2}$$

Notice, that several $Z \in \mathcal{X}$ with $\exists! \mathbf{y} = (y^1, \ldots, y^n)$, $y^i = f(Z, \cdot)$, $n \geq 2$ can exist (Z with multiple applications).

Definition 2.3. The corresponding set of applications of X is

$$Y(X) = \{y \mid y = f(Z, \cdot), Z \in \mathcal{X}(X)\}. \tag{3}$$

As mentioned above, $y = f(Z, \cdot)$ indicates, that Z might be but does not have to be combined with other code. And, of course, it is also possible to replace code lines. Thus, a general notation is

$$y = f(Z, \cdot) \in \{f(Z), f(V)\} \text{ with } V = \{\{Z \cup W\} \setminus U\} \neq \emptyset, Z \in \mathcal{X}(X), \quad (4)$$

which yields

$$Y(X) = \{y \mid y \in \{f(Z), f(V)\}\} = \{f(Z), f(V)\} = Y(Z) \cup Y(V). \quad (5)$$

2.2 Definition of Scarce, Non- and Anti-Scarce

A *scarce* ressource is a ressource with *rivalry in use*. Thus, the resource X is called a *scarce resource* with respect to $\breve{Y} \subseteq Y$, if the use of $Z \in \mathcal{X}$ for application $\breve{y} \in \breve{Y}$ leads to *rivalry in use*, i.e.

$$\forall \breve{y} = f(Z, \cdot) \mid [X^{new} = \{X \setminus Z\} \subseteq X] \wedge [Y^{new}(X^{new}) \subset Y(X)] \quad (6)$$

A *non-scarce* ressource is a ressource with *non-rivalry in use*, this refers to public and club/toll goods. Thus, the resource X is called a *non-scarce resource* with respect to $\tilde{Y} \subseteq Y$, if the use of $Z \in \mathcal{X}$ for application $\tilde{y} \in \tilde{Y}$ leads to *no rivalry in use*, i.e.

$$\forall \tilde{y} = f(Z, \cdot) \in \tilde{Y} \mid [X^{new} = X] \wedge [Y^{new}(X^{new}) = Y(X)] \quad (7)$$

An *anti-scarce* ressource is a ressource with *anti-rivalry in use*, i.e. the more the ressource is used, the higher is the value of the ressource, and/or the ressource itself increases due to use. Thus, the resource X is called a *anti-scarce resource* with respect to $\hat{Y} \subseteq Y$, if the use of $Z \in \mathcal{X}$ for application $\hat{y} \in \hat{Y}$ leads to *anti-rivalry in use*, i.e.

$$\forall \hat{y} = f(Z, \cdot) \in \hat{Y} \mid [X^{new} \supseteq X] \wedge [Y^{new}(X^{new}) \supset Y(X)] \quad (8)$$

2.3 Modeling Property Rights

In a world with rational individuals having complete information and knowledge, there would be bargaining and trade of each $y \in Y$. Thus, transaction costs, and incomplete information/knowledge, give reason to the fact that

property rights are defined and traded. Hence, the theoretical framework has to contain a consistent modeling of (intellectual) property rights (as this paper is on software, in the following I always refer to IPRs only, although in principle the model can be applied to PRs and IPRs.)

Following e.g. Furubotn & Richter (2005), Eggertsson (1990), Hart & Moore (1990), I broadly distinguish between the set of coordination rights (usus and abusus) from the residual rights (usus fructus and alienation rights). The complete set of rights is therefore defined as $H = \{H^c \cup H^r\}$, with H^c as the set of coordination rights, and H^r as the set of residual rights. Let $h \in H$ denote one property right and $Y' \subset Y$ the set of *known* applications of X.

At first, coordination rights are defined: The conjunction of a coordination right $h^c \in H^c$ with a resource X ($h^c : X$) leads to a distinction-criteria between the applications that are covered by the IPR and those which are not. Notice that there is no need to know the whole set of possible applications, as the distinction-criteria is like a selecting-rule r which tells one whether any $y \in Y(X)$ is covered by the IPR or not: $r : y \rightarrow [0,1] \ \forall\, y \in Y$. This yields

$$h^c : X \longrightarrow \{y \in Y' \mid r(y) = 1\}. \tag{9}$$

For example, let $\mathbf{h}_u$ denote the vector of all usus rights, $\mathbf{h}_a$ the vector of all abusus rights, and $\mathbf{h}_{u\&a}$ all usus and abusus rights. This yields e.g.

$$\mathbf{h}_u : X \longrightarrow \{y \in Y' \mid y = f(Z)\} = \{Y(Z)\}, \tag{10}$$

$$\mathbf{h}_{u\&a} : X \longrightarrow \{y \in Y' \mid y = f(V)\} = \{Y(V)\}. \tag{11}$$

Next step is to define the residual rights, where I have to distinguish between usus fructus and alienation rights: Let $\mathbf{h}_f$ denote the vector of all usus fructus rights, and π the payoff gained from y, then

$$\mathbf{h}_f : X \longrightarrow \{\pi \mid \pi = f(y),\ y \in Y'\}. \tag{12}$$

The right to transfer IPRs of a resource has to be modeled in a slightly different way, as it is in a sense a 'right on rights'. Let h^h denote the alienation right regarding h. An individual holding an alienation right with respect to

h can therefore *decide* wether to keep on holding this right, or transfer it, i.e. do not hold it anymore:

$$h^h = \text{'decision'} \circ h = \begin{cases} 0 & \text{if right } h \text{ is transferred,} \\ h & \text{if right } h \text{ is not transferred.} \end{cases} \tag{13}$$

Let $\mathbf{h}^h$ be the vector of all alienation rights. Holding alienation rights on a resource X is then formally given by

$$\mathbf{h}^h : X = \begin{pmatrix} h^{h^1} : X \\ \vdots \\ h^{h^n} : X \end{pmatrix} = \begin{pmatrix} [0,1] \cdot h^1 : X \\ \vdots \\ [0,1] \cdot h^n : X \end{pmatrix}. \tag{14}$$

Notice, that there exist "recursive" alienation rights. A recursive alienation right is the right to transfer an alienation right. This means, that

$$\exists\, h^k \in \{h^1 \dots h^n\} \text{ s.t. } \left[h^k = h^{h^j} \text{ with } h^j \in \{h^1 \dots h^n\},\, h^j \notin \mathbf{h}^h \right]. \tag{15}$$

3 The Resource Software

3.1 About the Economic Characteristics of Software

Software is a good with particular economic characteristics leading to specific traits of software markets regarding e.g. the cost function, the question of compatibility and IPRs. To simplify, one can subsume the economic properties of software as follows (for more details see von Engelhardt 2006):

Software is a *digital good* and therefore recombinant: software products are "cumulative and emergent—new digital goods that arise from merging antecedents have features absent from the original, parent digital goods" (Quah 2003, p 19) which tends to result in economies of scope. Within this text, the terms *recombinable*, *combinable* and *cumulative* are defined as follows: Let $\mathcal{S}(X)$ denote the set of all permutations of X, and $S \in \mathcal{S}(X)$ denote one permutation of X. X is called *recombinable* if

$$\exists\, S \neq X \text{ s.t. } \exists\, y = f(S). \tag{16}$$

This means, it is possible to derive from one given source code at least one new recombinant source code just by rearranging the code lines. This is more or less a theoretical eventuality only, more likely (a part of) a source code is combined with other/new code lines. $Z \in \mathcal{X}$ is called *combinable* with respect to W, if

$$\exists\, W \neq \emptyset \text{ s.t. } [\exists\, y = f(V),\, V = \{\{Z \cup W\} \setminus U\} \neq \emptyset]\,. \tag{17}$$

Software is aspatial, thus it is infinitely expansible and therefore nonrival: once software is produced, it can be reproduced without any loss of quality and the reproduction costs are virtually zero. But there are high development and pre-launch testing costs (high first copy costs). These high sunk costs combined with the low marginal costs lead to a subadditive cost function (economies of scope do not necessarily support this, see Baumol 1977).

Software is a *network good* with direct and indirect network effects. As software is data processing, there is an exchange of data. This exchange happens either with other software (applications and/or operating system) or hardware or both, which requires compatibility. This implies that producers have an incentive either to use the dominant standard or try to push their own standard. Consequently a coordination problem arises: which standard should be used, will it be proprietary or open? On the supply and the demand side two different forms of network effects play a role. The first is the so-called installed base effect, i.e. the utility increases with the total number of users (producers and consumers). Second, there is a personal network effect (Westarp 2003), where the adoption decision is determined less by the total numbers of users but by the adoption decisions in the personal network. However, network effects are intimately connected with *complementarity* which "means that consumers in these markets are shopping for *systems* [...] rather than individual products" (Shy 2001, p 2, emph. in original), and *modularity* (even of parts of software) plays an important role (Weber 2004, pp 172 ff; Langlois 2002, pp 22 f). A necessary condition for 'complementarity' and 'modularity' is *compatibility*.

This leads to another characteristic of software, for which *compatibility* and *combinability* are necessary (but not sufficient) conditions: If X is combinable and V is compatible to X, then X can be cumulative. X is cumulative, if V itself is part of the new X^{new}, i.e. X is called *cumulative* with

respect to W, **if** $\exists\, W \neq \emptyset$ *such that*

$$[\exists y = f(V),\, V = \{\{Z \cup W\} \setminus U\} \neq \emptyset] \wedge [Z, V \subseteq X^{new}] \tag{18}$$

Software is an *information good* because the source code (the system of algorithms) is information, i.e. a human-readable recipe: software is a system of instructions for the data processing. Thus, every kind of software can be described as algorithms forming a logical construction, build to perform one or several task(s). While distinguishing information from knowledge Dosi points out that algorithms are information (Dosi 1996, p 84). But software differs from other information goods, as the average consumer does not care about the information but merely about the impact (for example if one uses an email-client program, one wants to receive and send emails but (mostly) does not want to read the source code). This explains why software is an information good that can be sold in a state users can not read the information: proprietary software is typically given away only in state of (only machine-readable) binary codes and therefore the information is 'closed'. Hence within this paper the term closed source software (CSS) is used instead of 'proprietary software'.

3.2 Software as a Non- and Anti-Scarce Ressource

Due to the characteristics of software, namely being recombinant and aspatial, a source code is a *non-scarce ressource*. Obviously there is no rivalry in consumption, but there is also no rivalry in production:

(i) Because of virtual zero reproduction costs, a given first copy X is a non-scarce resource for producing $n+1$ copies. X can be copied without any loss of quality, thus $y :=$ 'copying' $\mid [X^{new} = X] \wedge [Y^{new} = Y]$.

(ii) Because of lack of physical abrasion and being combinable, a given source code X is a non-scarce resource for further software development. X can be used (even in parts) as input stock in developing first-copies of new, derived software product:[4] $y = f(V) \mid [X^{new} = X] \wedge$

[4] Because of this there are whole catalogs of complete elements of programs (Gröhn 1999, p 5) and a distinct programming approach—the so-called component-based software engineering—emerged. This approach also includes the re-use of software components across producers (Romberg 2003, pp 253 ff.).

$[Y^{new} = Y]$.

(iii) Because of software is an information good, a given source code X is a non-scarce resource for transfer of ideas and learning, thus knowledge spillover. A source code can be interpreted as a list of programming solutions. For a new software project, several solutions provided by existing source code might be useful and can be used in sense of a transfer of ideas and concepts even from one programming language to another. But also without an analogous transfer of ideas, reading a source code of interest can be beneficial for a software engineer, as one can learn from it and thereby improve programming skills. Hence, for all 'applications' y that are transfer of ideas or learning, applies $y = f(Z) \mid [X^{new} = X] \wedge [Y^{new} = Y]$.

Thus, a source code is a non-scarce ressource. Furthermore, software is an 'at least non-scarce ressource', as there is not only non-rivalry in use, but to some extent even anti-rivalry[5] in use, hence it is a *anti-scarce ressource*

(iv) Because of the importance of standards and network effects, software is a network good with respect to the supply side as well as to the demand side (e.g. see White et al. 2004, Kooths et al. 2003, Gröhn 1999, Gandal 1994). If X is a network good regarding a set of 'network applications' $y^{nw} \in Y$, then X is an anti-scarce resource with respect to y^{nw}:

$$\forall y^{nw} \mid [Y^{new} \supset Y] \wedge [X^{new} \supseteq X] \qquad \text{cf. (8)}$$

This is true for all network goods, network effects respectively. One intuitive example is a network of telephone wires: The more users plug in to the network, the more applications ('call person A') are possible.

(v) Because of cumulativenes, there is anti rivalry in use of a source code: If X is cumulative with respect to any $W \in \mathcal{W}$, $V = \{\{Z \cup W\} \setminus U\}$, then X is an anti-scarce resource with respect to $\mathcal{W}$:

$$\forall W \in \mathcal{W} \mid \{[\exists y = f(V)] \wedge [Z, V \subseteq X^{new}]\} \qquad \text{by (18)}$$

[5]Compare the following also with Weber (2004, pp 153 ff.), where one can find similar thoughts, Weber uses the term 'antirivalness'.

$$\Rightarrow \left[(V \not\subseteq X) \wedge (V \subseteq X^{new})\right] \wedge \left[(y \notin Y) \wedge (y \in Y^{new})\right]$$

$$\Rightarrow \forall y = f(V) \mid [X^{new} \supset X] \wedge [Y^{new}(X^{new}) \supset Y(X)] \qquad \text{cf. (8)}$$

If a software engineer further develops a module—e.g. because of own needs—others can benefit from this improvement as long as the new piece of source code is still compatible and the new 'module' is implemented in, and improves users' software systems. Thus, a source code is a potential input stock for further software development, where the new output is (potentially) another incorporable module for the software system and at the same time is again (potential) input stock that lowers further software development costs, where the result again can be implemented, and so on.

4 Copyright and the Non- and Anti-Scarce Resource Software

The Coase Theorem (Coase 1960) tells, that if transaction costs are zero, external effects are perfectly internalized and the allocation of resources does not depend on the distribution of property rights (PRs). Hence, in a world *with* transaction costs PRs—just as any institution—'matter', and the role of PRs is to internalize external effects as good as possible. Hence this section is about the role of (I)PRs with respect to a non- and anti-scarce resource. As the paper focuses on the resource software (source code), this section is about the role of copyright based licenses, because software is traditionally protected by copyright law (Graham & Somaya 2004, p 269). and the software license agreements define the transfer of the rights.

4.1 Optimal Allocation and Optimal Licenses

In this section I argue, that there exists an optimal allocation of rights regarding non- and anti-rival applications, and optimal licensees can be defined:

As software has a subadditive production function, the provision of software is an example of a so-called 'natural monopoly' (Baumol 1977). With respect to the non-rival applications it is about a private provision of a good

without rivalry in use, hence a club good story. It is well known from standard micro-economics, that a commodity without rivalry in use should be supplied, if the sum over each individual's willingness pays (WTP) at least covers the total costs (C), hence if $\sum_{j=1}^{m} WTP_j \geq C$, with m agents.

Let's assume, that there is a finite number of applications $y^i \in Y$, $i = [1 \dots n]$, and each of the n applications is traded in one market each. It is also assumed, that

(i) the owner of the resource can price discriminate, such that each agent pays an individual price for the application i denoted by p_j^i

and

(ii) each of the n-markets is a contestable market, such that the incumbent has to choose the lowest price-vector covering the costs.[6]

This yields

$$\sum_{i=1}^{m} p_j^i = C^i, \;\; p_j^i \leq WTP_j^i, \qquad \forall i = [1 \dots n] \; \forall j = [1 \dots m] \tag{19}$$

with C^i is the portion of y^i of the (First-Copy) costs of Y. Obviously this is a welfare maximizing private provision of the resource.

Additionally, anti-rival applications—the rights to use the anti-rival applications respectively—can be allocated optimal as well: It is known from network theory, that ownership can internalize network effects, because if it is possible to price every single plug-in, the network effects can perfectly be internalized by dynamic, i.e. individual price discrimination that takes into account the marginal benefits of adoption (Liebowitz & Margolis 2002, 1994, Katz & Shapiro 1994). Thus, optimal internalization requires perfect price discrimination with respect to adoption, hence each adoption must be traded separately. This can be applied to *any kind* of *positive feedback* mechanism in

[6] Of course, assuming perfect contestable markets in context of software seems to be a quite problematic assumption. But the argument does not change, if one allows market power: in this case the monopolist is able to gain an extra profit, but as the monopolist is doing perfect price discrimination, welfare is maximized also in this case. Hence the contestable-market-assumption is not a critical one in the context of the argument, but it simplifies the argument.

use. Thus, regarding to the model of this section, it is again sufficient, that the source code owner can perfectly price discriminate, the result is a welfare maximizing allocation.

Hence, there are n markets where the y^is are traded, and m agents paying a price $p_j^i \geq 0$ for each $y^i \in Y$ (with $p_j^i = 0$ implies that agent j has a zero *WTP* for application i and hence does not buy it at all). Therefore one can represent the whole economy with the payment matrix P given by

$$P = \begin{pmatrix} p_1^1 & \cdots & p_1^n \\ \vdots & \ddots & \vdots \\ p_m^1 & \cdots & p_m^n \end{pmatrix}, \tag{20}$$

with each market represented by a column. The payment structure of an agent is given by the corresponding row and can be written as $\mathbf{p}_j = (p_j^1, \ldots, p_j^n)$

Additionally, one can represent the complete allocation with one single matrix: Let A be a $n \times m$ matrix with $a_i^j = y_i^j$. Due to (19) we have $y_i^j = 1 \iff p_i^j > 0$ and $y_i^j = 0$ else. For example in case of $n = 5$ and $m = 4$ one possible allocation is given by

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & o \end{pmatrix}. \tag{21}$$

Assumed, that it makes sense to write a license agreement that covers a set of applications instead of trading each application separately, then this license agreement is optimal if it transfers PRs to agent i such that all applications are covered the agent would pay a positive price for:

$$\mathbf{h}_j : X \to \{y^i \mid p_j^i > 0\}. \tag{22}$$

The price for this license agreement is given by

$$p_j = \sum_{i=1}^{n} p_j^i = (p_j^1, \ldots, p_j^n) \cdot \mathbf{1}^T, \tag{23}$$

with $\mathbf{1}^T$ is the transpose of the one-vector.

To sum up: One can define the welfare maximal allocation of non- and anti-rival applications, and a perfect market with perfect price-discrimination would lead to this allocation. Based on this, optimal defined, copyright based licensee agreements can be defined, hence an optimal allocation of PRs is theoretically possible as well.

4.2 Ex Ante Transaction Costs and Incomplete Knowledge

As mentioned above, transaction costs and incomplete knowledge gives reason to the fact, that PRs are traded. In this section it is argued, that neither limits in market trading because of ex ante transaction costs, nor incomplete knowledge can reason a crucial internalization problem such that governance structures have to be build.

4.2.1 Incomplete Knowledge

Incomplete knowledge gives a good reason why PRs are defined and traded (see also p 7), because if agents only know $Y_i' \subset Y$ it makes sense to trade 'rules' rather than applications. Regarding the question of crucial limits of market allocation of optimal defined PRs, the important point is, that incomplete knowledge changes the standard market game into a Bayesian market game, but nothing more. This is now a case of Bayesian rationality with the agents defining, trading and pricing the rights optimally based on their subjective expectations regarding Y. Without going into details regarding the specific characteristics of markets with Bayesian probabilities, one can conclude, that incomplete knowledge is no good reason to state a fundamental problem with the market based allocation of PRs. Additionally, one of the basic functions of markets is to *create new knowledge* (Hayek) by rewarding innovative use of resources, and PRs enable innovators to appropriate such returns.

4.2.2 Ex Ante Transaction Costs and Limits of Internalizability

Next argument could be, that ex ante transaction costs (search and bargaining costs) inhibit optimal internalization via proper defined PRs, as there is a lack of internalizability:

Whereas a new network member causes an (more or less countable) increase of the value of the network only once and exactly at the moment of plug-in, in order to internalize the positive effects caused by the the cumulative effects described in item (v) one would have to be able to provide the engineer a benefit each time the new source code—the new module—is incorporated in someone's software system. Obviously, transaction costs inhibit such an internalization, especially when it is about only (very) small steps in improvement and little changes, thus small step cumulative development. In the case of knowledge spillovers described in (iii), it is even more obvious, that neither the point of time nor the frequency of future value creation is known. The moment of 'adoption' that causes knowledge spillovers is hard to observe and moreover the value of the 'adoption' is hard to evaluate as the results are increased skills that might lead to better future performance.

Hence, the economic value of this increased performance as well as future cumulative effects are obviously hard to measure ex ante, especially in the context of innovations, because "as Arrow himself pointed out long ago, if an innovation is truly an innovation it is impossible for a finite observer to precisely forecast it." (Dosi 1996, p 84). However, one can again argue, that it is sufficient if individuals are able to form subjective expectations regarding the future benefit of the knowledge, just like in case of a reference book or textbook for example.

Thus, there might be a lack of internalization because of lack of internalizability. This lack of internalization, thus the existence of positive *externalities* leads to too little of the relevant activity as the social benefit is greater than the private benefit. That means, if IPRs fail to some extent to internalize external effects with respect to source code, one can postulate market failure to some extent regarding this issue. But the degree of this 'market failure' is to some extend limited.

Anyhow, there is another problem: In order to be able in general to internalize the external effects, IPRs must be able to fence, i.e. to corral the effect, in order to be able to control. But regarding the non-internalizable positive external effects discussed in this paper, the corralling effect becomes a problem: Regarding some kinds of positive externalities, due to transaction costs, IPRs like copyrights do not only fail to internalize but can also inhibit those positive effects, if they fence it. The latter gives (utilitarian) theoretical justification for governmental limitation of the scope of IPRs: From a

natural right philosophy point of view—most prominently represented by Locke (1698) and Kant (1798)[7]—one must give good reason why laws limit the scope of intellectual property, thus why intellectual property law grants protection only under certain conditions, that is to say the disclosure of the underlying idea (or: information) and the termination of the exclusive rights after a certain period of time.

However, this does not necessarily contradict individuals interest, as as it might not cause real costs for the source code owner to forgoes such rights: The costs of not internalized positive effects are costs in sense of missing revenue, therefore in the following they will be called 'indirect costs'. Let $Y_i^{tc} \subset Y'$ denote the set of applications an agent i can realize and/or trade, can trade the corresponding IPRs respectively. The superscript tc indicates, that this set is determined by transaction costs while the subscript i indicates that this set also depends on individual 'factors', e.g. the access to necessary resources, etc. To give an example: a student's Y_i^{tc} of a certain source code might be smaller than Microsoft's Y_i^{tc} of the same source code.

First, the role of IPRs regarding non-scarce resources is defined: Assuming that envy and (irrational) stinginess does not play a role, there is no reason why an agent i should claim and/or not freely transfer the rights $\mathbf{h}$ that cause *no* indirect costs:

$$\mathbf{h}: X \longrightarrow \left\{ \tilde{y}^o \notin \tilde{Y}_i^{tc} \mid (\nexists y^{tc} \in Y_i^{tc} \mid \varepsilon_{tc,o} \leq 0) \right\} \tag{24}$$

with $\varepsilon_{tc,o}$ as the cross-price elasticity of $y^{tc} \in Y_i^{tc}$ regarding $\tilde{y}^o$.

This leads to the definition of the *optimal* IPRs with respect to a non-scarce resource. The rights $\mathbf{h}$ that should be claimed is given by

$$\mathbf{h}: X \longrightarrow \left\{ \left\{ \tilde{y}^{tc} \in \tilde{Y}_i^{tc} \right\} \cup \left\{ \tilde{y}^o \notin \tilde{Y}_i^{tc} \mid (\exists y^{tc} \in Y_i^{tc} \mid \varepsilon_{tc,o} > 0) \right\} \right\}. \tag{25}$$

Thus, IPRs regarding a non-scarce resource are defined in an optimal way, if they protect (i) all applications agent i can *realize and/or trade*, and (ii) all applications that are *substitutes* to any $y \in Y_i^{tc}$

Regarding the feedback-effects, the argument is basically the same. Additionally one has to take into account, that it can be rational not to claim ex-

[7] For a brief overview of utilitarian and non-utilitarian philosophical foundations of IPRs see Menell (2000).

clusive PRs regarding some $\left\{\hat{y}^o \notin \hat{Y}_i^{tc} \mid (\exists y^{tc} \in Y_i^{tc} \mid \varepsilon_{tc,o} > 0)\right\}$ and even regarding some $\hat{y} \in Y_i^{tc}$ if the benefits from the feedback effects (over)compensate the costs. I will come back to this later.

Copyright is defined and works in a way, that protects the tradeable, hence private internalizeable, effects, and forgoes the right for such positive effects, that are not internalizeable. Doing so, it combines private (ex ante) incentives to produce with the disclosure of the information (ex post efficiency)[8], as copyright does not protect the idea itself—the pure information—but its expression. Thus, in the case of e.g. a copyright protected book the author earns money from its publication, which is a disclosure of the ideas (or: information). Thus, with increasing sales figures, the author earns more money and the ideas of—the information within—that book diffuse, because everybody who buys that book can read it.

Traditionally software is protected by copyright, but, when thousands of hundreds of copies of copyright-protected proprietary software are sold, no one gets the information, there is no information diffusion—because the software is given away only as binary code. Since software is a exceptional information good, IPRs defined by copyright law are not able to relieve the tension between ex post efficiency and ex ante incentive:

- If software should be sold for a positive price, it generally[9] has to be given away in the state of binary codes. This is the case of proprietary or closed source software (CSS): there is no disclosure and no diffusion of the information. But due to the price greater than zero the ex ante incentive condition is fulfilled. Obviously, the information has to be hidden in this case, one has to protect the source code in order to sell it

[8] In general, IPRs are designed, to find a balance between the positive and negative effects of strong IPRs, that is to combine incentives for individuals to produce—the ex ante incentive—with the disclosure of the information—the ex post efficiency (Cowan & Harison 2001, Quah 2003, pp 16 f, 19 ff). Hence, patents do not protect the idea itself but its application in form of machine, method or matter (patent) (Besen & Raskind 1991, p 12). The right to be a temporary monopolist regarding the economic use of a novel technical solution is bundled with the constraint to disclosure the information that stands behind the innovation, as the technological solution has to be described in the patent specification. The case of copyright is discussed in the text above.

[9] There exists some exceptions from this, especially when it is about high-specific single contracts. But there are even some examples where the source code is delivered by default like in case of the SAP R3 license (Böhnlein 2003, p 23).

for a positive price, because if the information/the source code would be disclosed, the price would be zero.

- Contrary, to reach ex post efficiency the information has to be disclosed. But, as already mentioned above, free access to the source code in practice implies a price equal to zero. This is the case of the so called open source software[10] (OSS): The source code—the information—of open source software is 'open', therefore disclosed. If OSS is sold for a positive price, it is always bundled with complementary goods like hardware or service, OSS alone is always available for free (For more details on those business models see e.g. Dahlander & Magnusson 2006, Brügge et al. 2004, Leiteritz 2004).

Another aspect regarding IPRs and software is the discussion about the so-called software patents (e.g. see Blind et al. 2005, Bessen & Hunt 2004*a*,*b*, Mundhenke 2004, Gallini 2002, Canfield 2006). But as the two production modes open vs. closed source software exist in countries with and without software patents, and because both are based on copyright law, this paper focuses on copyright issues of software.

In addition to the well known result, that not (sufficiently) internalized positive external effects lead to an underprovision, in case of source code, the lack of internalizability leads to a too small group of beneficiaries as the positive effects are "corralled", and therefore to some extend inhibited. This concerns knowledge spillovers as well as to cumulative activities. The reason for this—to some extend already mentioned above—will be discussed in the next section: the problem of not exclusively separable rights.

4.3 Ex Post Transaction Costs and the Problem of Not Exclusively Separable Rights

Internalization of the positive externalities through ownership requires, that the IPRs regarding software can corral (fence) the positive effects. In this case, there is excludability combined with no rivalry in use of the positive

[10] The term 'free/libre open source software (FLOSS) could be used instead, but as this paper wants to point out the differences resulting from different use of intellectual property law, terms 'open' source vs. 'closed' source software are preferred.

externalities. This case refers to club or toll good: anyone who wants to participate has to join the club, and therefore has to pay a toll.

Obviously it makes no sense to claim non-enforceable IPRs, but it also makes no sense to transfer rights, that are not exclusively separable: I call a right a exclusively separable right, if the right holder is *de facto* only able to do, what is covered by the right. Examples with respect to software are the right to use a software without changing the source code and the right to copy a software, as usually the transfer of such rights implies that the software is given away only in state of binary codes, and in case of use only, there is technical copy protection in addition. Hence, who ever receives such rights is—at least to some extend—de facto *not able* to do things that are not covered by the rights because of technical reasons. Thus, IPRs with respect to software *are* exclusively separable at least if the applications covered by the right do not imply the need for access to the source code.

In principle, there exist exclusively separable IPRs regarding applications described in item (i) and (iv) (section 3.2 pp 10,11) as well as consumption i.e. use software without changing the source code. Contrary to this, rights regarding (iii) (knowledge spillover) and (v) (cumulativenes) are in principle not exclusively separable. The applications of item (ii) are in between: the existence of organizations like 'code-sell.com' proves that selling source code is possible, although transaction costs[11] might impede some market transactions. However, the problem of defining exclusively separable IPRs with respect to (iii) and (v) is, that such applications need access to the source code. This implies the following: If one wants to trade rights that are optimally designed, one has to grant access to the source code. This leads in a sense to a *club of source code users*, and and this might affect other rights, as problems with shirking and misuse can occur. The term *shirking* refers to a performance that is lower than the agreed effort, thus, this refers to prob-

[11] This refers to transaction costs of the market in general but especially to problems of asymmetric information, namely of hidden characteristics (Arrow's information paradox) and hidden action: The First refers to, that whenever information is to be traded, the so-called information paradox occurs, as the buyer of information is not able to determine its value before the transaction unless the vendor reveals the information. However, if the information is revealed, there is no longer a motivation nor a necessity to actually purchase the information. The latter is about the monitoring and control costs regarding the question of unauthorized re-sell of information, source code respectively.

lems covered by e.g. principal agent theory or theories on team production and collective action problems. The term *misuse* refers to the problem, that the source code is used in a way that is not covered by the contract.

Transaction costs can cause shirking and misuse, thus may inhibit reaching the optimal club size. A club owner normally offers only the usus of the club-good, and often keeps a right of expulsion, that is to fetch back the usus from members who did not comply with the rules But in our case there is a de facto 'dilution of the property rights' (Picot et al. 2005, p 47): Although the original owner of the source code might be still the formal owner—i.e. formally still exclusively holds abusus, usus fructus, alienation right and the right of expulsion—but in practical there is a dilution of the IPRs, because of transaction costs: An increase of club members induce controls costs, thus with a huge amount of members it is simply not possible anymore to control if some members re-use the source code for own purpose and/or re-sell the code. The latter gives reason to the fact, that a right to expulse is not enforceable in groups with (too) many members. Hence, with an *increase* of 'cooperation-club' members c. p. the feasibility of shirking *increases* as policing and enforcement costs *increase*. Figure 1 depicts this problem: With increasing number of members, transaction costs (TC) increase. Assumed, that the owner of the resource would first trades with the agent offering the highest price, the m agents are arranged by the price they are willing to pay. Hence, the returns as a function of club members $R = f(m)$ is concave. The optimal number of members is indicated by the dotted line. It is easy to see, that if transaction costs are high, the optimal number is small, (maybe zero).

Some policing and enforcement costs as well as bargaining and decision costs c. p. *decrease* with the *decrease* of specificity of the cooperation contract, but a *decrease* of specificity c. p. *increases* feasibility of shirking. The first part of this logic is shown in figure 2: the transaction costs decrease with decrease of specificity i.e. the increase of set of allowed applications, simply because if one allows every possible application one does not have to control anything. Additionally the price one can get from selling somebody a licensee agreement would be maximal if this licensee would allow every possible application. (Notice, that for the argument it is irrelevant whether one assumes that the transaction cost curve is inverse U-shaped or strictly decreasing.) Of course, a *decrease* of specificity c. p. *increases* feasibility of shirking. But the set of applications an agents de facto can use, and hence the price an agent

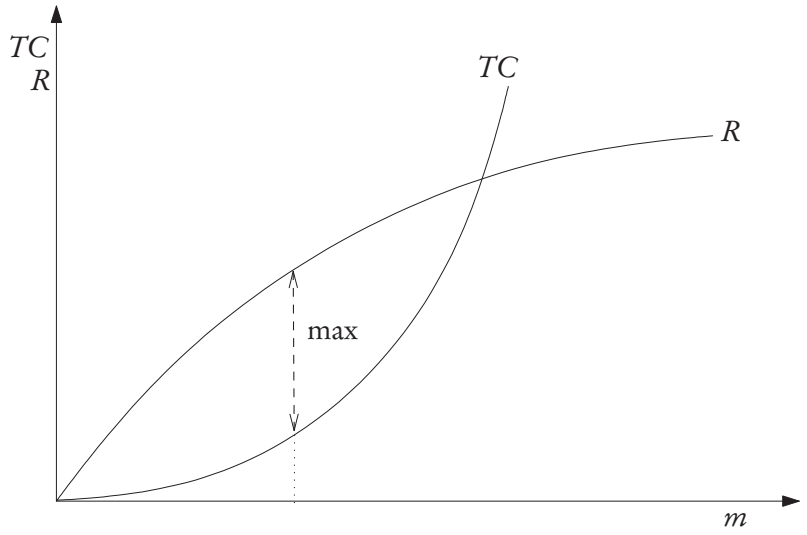

Figure 1: Transaction Costs and Limits of Size (Members)

will be willing to pay for an complete unspecific contract, is determined by the agent's set of realizeable applications (Y_i^{tc}). This implies, if the resource owner would *know* each Y_i^{tc}, optimal PR-allocation would be reached again.

Obviously the problem is that the agents do not know other agent's Y_i^{tc}. Hence, they do not know the *type* of the other agents, and this leads to a problem well known as 'adverse selection' (because Y_i^{tc} determines ex post (hidden) action of agent i, the ex post problem can be transferred to an ex ante information problem): As there are incentives to indicate a smaller Y_i^{tc} than the real one, because this would lead to a smaller price, the agents will not truly indicate their Y_i^{tc}. The owner of the source code might know—or has a sufficiently correct idea about—the distribution of Y_i^{tc}, the average set of realizeable and tradeable applications $\bar{Y}_i^{tc}$ respectively. But given the corresponding average price, at least some agents with $Y_i^{tc} < \bar{Y}_i^{tc}$ won't pay for this, and leave the market in a sense. This increases the average set of applications and therefore the average price, and so on, and so on. This yields the well-known result of adverse selection in insurance markets: At the end,

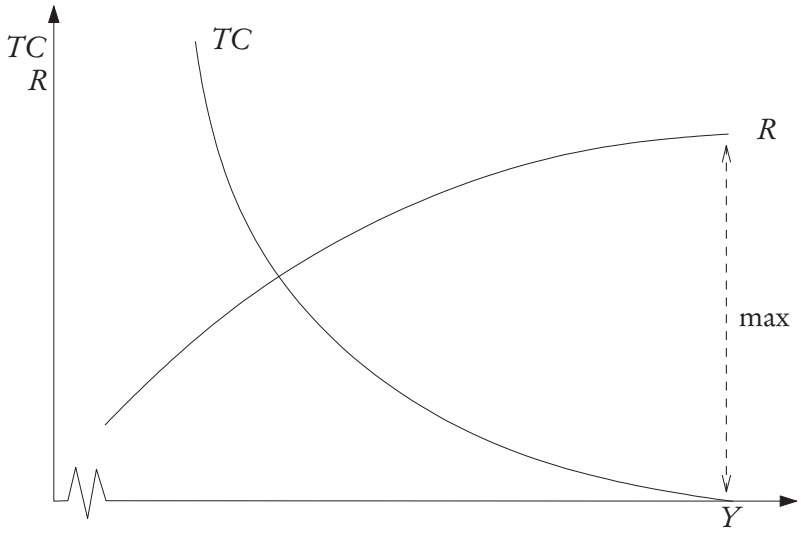

Figure 2: Transaction Costs and Specification of Contract

only the agent with the largest set of applications will rest in the market.

Although this version of the paper does not yet provide a complete analytical model of the limits of market trading of not exclusively separable rights, the intuition was explained: Because of ex post transaction costs, PRs that imply access to the source code are de facto not exclusively separable. This limits the market allocation of the PRs. Next step is to ask, what kind of governance structures can be build, as answer to the problem of dilution of exclusive ownerhip because of the problem of not exclusively separable rights. The following section is about this.

5 Two Co-Existing Production Modes: OSS vs. CSS

Based on the analysis above, this section focuses on the principles of CSS and OSS each, but first describes the different licenses.

5.1 CSS vs. OSS Licenses

The existing software licenses can be classified by the scope of transferred rights (for the following see Hawkins 2004, p 107; Böhnlein, 2003, pp 19 ff; Nüttgens & Tesei, 2000, p 11):

	usus	usus fructus	abusus	alienation right
CSS	+	+	-	(-)
OSS (viral license)	+	+	+	(+)
OSS (public license)	+	+	+	+

Table 1: The Transfer of Usus, Usus Fructus, Abusus and Alienation Right

- CSS licenses are *exclusive* as they are based on the principle of closeness, i.e. a user (licensee) of CSS typically receives only the usus and (maybe restricted) usus fructus from the licenser, and the alienation right is not transferred or restricted (see table 1).

- OSS licenses are *inclusive* as they are based on the principle of openness, i.e. the licenser offers his license to anybody who wants it and OSS licenses *in principle* transfer the whole set of rights, although they differ in the scope of transferred rights: *Public* OSS licenses—like the BSD license—do not restrict the use of the software and the software's source code in any way, whereas *viral* licenses—like the GPL—differ in the alienation rights, as the right to redistribute is restricted: Any further developed software as well any derived work must be licensed as a whole under the same OSS license (see table 1). Hence, OSS is not software without any property, as e.g. the "GPL contains provisions covering property rights and licensing. It is based on copyright principles (...) [and] does not, as often misperceived, remove copyright protection" (Gehring 2006, pp 62, 70)[12]. One can interpret an OSS license as being a contract that offers everybody the whole set of rights

[12] Thus it is to some extend misleading, that the counterpiece of OSS *only* is called proprietary software, as 'proprietary' comes from the latin terms proprietarius and proprietas and its legal meaning is 'protected by copyrights'. Thus, some observers refer to the term closed source software (CSS) instead of proprietary software and so does the author of this paper.

while the possible constraint, thus possible limitation of the alienation right, must be considered only at the moment of redistribution.

The different kinds of software licenses fit the theory of not exclusively separable rights: CSS licenses, that are based on the principle of exclusive ownership, trade only such IPRs, that *are* exclusively separable, and do not allow access to the source code. OSS licenses in contrast offer the complete set of rights, as they allow access to the source code.

The software licenses reflect two different principles of OSS and CSS In the next two sections, this two different principles, i.e. the two different 'solutions', CSS and OSS are further examined.

5.2 The Principle of CSS

One solution to solve the 'dilution-of-control'-problem could be to increase control by adding a second set of rules, thus to build a governance structure that ensures control, i.e. build a 'firm'. With a software firm, the effects are internalized, as the firm owner does not only own X but also exclusively owns X^{new}. Additionally to the cumulativenes aspects, the benefits of knowledge spillovers are also internalized, at least as long the employee work for the firm. But building a firm is not costless and has its limits as one can not include everybody who would have been able to submit something. Thus, with respect to the maximal possible benefits from knowledge spillovers and cumulativenes effects, a firm can not reach optimal club size.

However, CSS is based on the principle to maximize exclusive rights. With exclusive rights it is possible to exclusively split coordination from residual rights, thus build a firm with a division of management and ownership. It is possible to produce software as a coordinated work of several software developers, who are employees, thus their role is defined by a employment contract and the employment contract contains a paragraph that makes sure, that all the possible copyrights are transferred to the company, thus at the end of the day they do not own any IPRs concerning the source code.

As developing software is more like finding solutions for a problem one may not be able to conclude directly from the output the effort of the developer. Software is therefore developed in a principal-agent-structure, as the principal defines certain aims and arranges a team of programmers, testers,

etc., but can barely monitor the effort and/or performance of these agents, because of monitoring costs (Pasche & von Engelhardt 2004, p 9), and well-designed contracts are needed to reduce inefficiencies derives from this. Additionally, of course, CSS firm can build cooperations to some extent, but as described above, there are some limits. However, the concept of CSS—i.e. the firm-solution—has its limit, i.e. can not reach the optimal level of internalization because in a world with transaction costs

(i) it is not possible to sell everybody the optimal individualized software that fits perfectly his or her needs.

(ii) it is not possible to write an employment contract with everybody who would have been able to do some cumulative activities.

(iii) firms are not costless. This refers to hierarchy costs, e.g. induced by the principal-agent problems described above.

To sum up: Firms are better than the market in solving the kind of control-problems caused by asymmetric information described in section 4.3, but they are not perfect in doing so, and are limited in their ability to create and internalize some of the positive effects.

5.3 The Principle of OSS

Another possible solution is the following: Depending on Y_i^{tc} it can be rational, to forgo exclusive rights regarding X, if this implies the opportunity to realize benefits from knowledge spillover and the cumulative feedback mechanism. I will focus on the latter mainly: Let's assume, that shirking can—at least to some extend—be avoided due to a mechanism that leads to a self interest in contributing i.e. cumulatively adding source code. In this case, a possible solution for the misuse problem is to allow more and combine this with passive control rights. If potential contributors have an self-interest in contributing, it is sufficient that a club owner has the right to decide whether an offered contribution will be included or not, in order to achieve high quality of the collective output. This means, production does not need 'active' control rights—that is to give someone the order to do something—but 'passive' control rights—that is the right to decide whether something will be

included or not—is sufficient. This means, the amount of exclusive IPRs can be minimized, thus a dilution of IPRs might play no role anymore.

As mentioned above, one can interpret OSS licenses as being quite general and 'open' contracts. Thus, OSS licenses are very non-specific cooperation contracts, designed in order to attract a large number of club members, and as OSS licenses are not limited in time, they seem to be designed for a co-operation with—at least potentially—infinite duration: At any time, anyone can access the source code and use it however long, given that the possible constraint, that is a possible limitation of the alienation right, is considered.

As OSS licenses are inclusive, with respect to the source code, it is not possible to hold exclusive rights, and therefore it is not possible to divide neither consumers from producers, nor software developers from coordinators or dividing coordination and ownership. Everybody involved in the (production) process holds the complete set of rights regarding the source code,[13] but the licenses are combined with institutions that support the ownership and the passive control rights of the different *projects:*

The different projects are open and permeable, but clear structured, thus they can be labeled as being passive hierarchical organizations, as the basic model of a OSS developing group is an example par excellance for coordination based on passive control rights: The basic organizational structure of such OSS-Projects is often labeled the 'onion layer' model (see e.g. Jensen & Scacchi 2007, Crowston et al. 2006, Wendel de Joode et al. 2003, pp 18,19): The first, outmost layer consists of passive users, while in the second layer one will find users who are more involved (writing bug-reports, testing new (pre-)releases), and so on. At the *core* are developers who contribute most of the code, have responsibilities and certain privileges. The *core developers* oversee the design and evolution of the project and control it. It exist clear rules about how one can move from the outermost layer into the core of the project, e.g. one have to prove software development skills, reliability etc. And, maybe most important, the core developers control—thus, manage—the project by using *passive control right*, thus their (exclusive) right to decide whether to accept or reject contributions (McGowan 2001, Wendel de

[13] In case of viral licensees there is of course a limitation of the alienation right, but this limitation affects everybody, thus all involved holds the same amount of limited alienation right.

Joode et al. 2003, p 20). The passive control rights are enforced by using the concept of ownership regarding the database in which the software is stored and the name—thus, the trademark[14]—of the project. This prevents cloning of projects and supports the signalling function of the project's name, thus trademark.

Thus, OSS projects combine non-specific contracts regarding the source code with *project*-clubs managed by passive control rights. The fact that OSS licenses are inclusive can be interpreted as an pragmatic reaction to the dilution of exclusive IPRs regarding the source code. Public licenses are the most radical in this context, whereas viral licenses limit the alienation right and thereby (passively) exclude at least those who do not want to license their products under such a license. But apart from that, contrary to the concept represented by CSS, OSS stands for the concept of minimizing exclusive rights. There is exclusive ownership only regarding the projects, that are managed by passive control rights leading to a kind of decision hierarchies.

Hence, OSS-projects are based on the principle of minimized exclusive ownership combined with passive control rights. This leads to two topics to be examined: Ho does this kind of passive control work? And: Why do economic agents have a self-interest in contributing to OSS-projects?

Given that OSS-projects have to compete with other projects as well as with CSS-products, they have to survive in innovate competition, hence they have to evolve sufficiently regarding quality and quantity (new features). This leads to the question of the optimal accept-reject decision: If the core developer's policy is to restrictive, then too little new source code is implemented, hence the project evolves too slowly. Additionally if too many improvements of too many contributors are rejected, the rejected code developers might have incentives to 'fork', i.e. to establish and develop their own version. But if the core developer's policy is to less restrictive, then the quality of the project growths to slow or even decrease. This kind of optimal accept-reject policy problem is in general an optimal level of exclusively question: like in any kind of exclusive clubs where the club members contribute something to the club, there has to be found the optimal level of restriction (neither a

[14] E.g. Apache is a trademark of The Apache Software Foundation, KDE and K Desktop Environment are trademarks of KDE e.V., Linux is a registered trademark of Linus Torvalds, and so on (see www.apache.org, www.kde.org, www.linuxmark.org).

club everybody can join, nor the most exclusive club no one can get in, is attractive).

Next question is, why do economic agents use OSS and contribute to OSS-projects. To some extend this question is answered by the literature dealing with theoretical and empirical research on the motives of open source software developers, as understanding OSS developing is to analyze the motives as well as the institutions. The motive related literature focuses more or less on the intrinsic and extrinsic motives of individuals to contribute in OSS projects. (An overview of this literature can be found in Rossi (2006, p 17 ff) Franck & Jungwirth (2002), Mustonen (2001), Johnson (2001), Schiff (2002) and Kooths et al. (2003); an additional online-collection can be found at opensource.mit.edu.). Authors like Dahlander & Magnusson (2006), Rossi & Bonaccorsi (2006), Lerner et al. (2006), Feller & Fitzgerald (2002), etc. discuss questions related to firms and open source software. Without going into details, one can mention, that on the one hand reasons can be found why firms *use* open source code, and on the other hand, why they *contribute*, which can be a firm's decision to publish (parts of) its source code as open source (e.g. Baake & Wichmann 2004), support—even substitute—open source software programs Mustonen (2005), engage in a project, etc.

The reasons why firms *use* open source software can be subsumed by the term 'ressources': using source code developed by others obviously reduces the own developing costs, and using a certain open source software to build a business model around can mean using a network standard for business. But this does not explain why firms contribute, as obviously the costs are minimized if they contribute nothing but use the given source code.

But why do individuals and firms have to *contribute* instead of just using given source code? Why is shirking so surprisingly seldom? *Contributing* can mean different things, from simply giving own developments to the community (on this, regarding embedded Linux, see Henkel 2006) up to getting involved in projects.

Besides motives that are more or less based on ideology ("OSS is a good thing, we have to support this" etc.)[15] one can give four reasons why even a

[15]It is questionable, whether ideological arguments/motives are stronger than 'selfish' motives in the context of OSS-based business models i.e. making money. See on this e.g. Rossi & Bonaccorsi (2006)

'selfish' economic agent contribute to OSS-projects:

(i) Contributing cause no real costs.

(ii) They are forced by the license and other institutions.

(iii) If reputation plays a role, then being part of a project is important.

(iv) Contributing source code is the only way to get at least some *control* over the further development of the project.

It is rational for economic agents to contribute to a project source code that they have developed anyway. They do so, as it causes no (or little) real costs for them to give this code away. Hence, (i) refers to that it causes no real costs to *forgo* the following rights: $\mathbf{h}: X \longrightarrow \left\{\hat{y}^o \notin \hat{Y}_i^{tc} \mid (\nexists y^{tc} \in Y_i^{tc} \mid \varepsilon_{tc,o} \leq 0)\right\}$.

Additionally, the institutions of OSS support—and to some even guarantee—the cumulative effect: Although one agent might be able to make money selling further developed source (hence, it causes significant (indirect) costs to give it away for free), licensees like the GPL force this agent to contribute his work back. Hence the license force the agent to forgo rights that belong to $\mathbf{h}: X \longrightarrow \left\{\left\{\hat{y}^{tc} \in \hat{Y}_i^{tc}\right\} \cup \left\{\hat{y}^o \notin \hat{Y}_i^{tc} \mid (\exists y^{tc} \in Y_i^{tc} \mid \varepsilon_{tc,o} > 0)\right\}\right\}$. In addition to this copyright-based institution, informal rules like the hacker ethic and community norms (incl. the enforcement characteristics) also support this cumulative effects, as at least some kind of contribution may be expected by the community. Thus, social norms can play a role here, as breaking the rules will be sanctioned by the community, that is stop cooperating or migrate to other projects (Osterloh et al. 2001, p 16 f). However, if individual benefits from using OSS (namely reduced developing costs) overcompensate the (indirect) costs, then OSS can make sense from an economic agent's perspective.

But remember, that the passive control game with the optimal accept-reject policy works only, if agents have a self-interest in being accepted. This leads to the question regarding the underlying logic, explaining why individuals as well as firms contribute *actively*. Only the last two points (iii) and (iv) can explain, why it is important for the agents to get their source code implemented, hence to be accepted:

Reputation in this context does not only refer to the 'career argument'—most prominently emphasized by Lerner & Tirole (2002)—but also to the

reputation of firms: It can be important that the firm contributed e.g. to the Linux-kernel, as this might be a signal for (a) quality skills as well as for (b) 'going with the community rules', if costumers prefere that: If it is known that the firm z plays an active role in a certain OSS project, this might increase the firm's ability to acquire customers.

The last point is 'control', an incentive not covered yet by the literature. The argument is as follows: If firms have a business model build on OSS, they want to sell complementary goods. Thus, they have a crucial self-interest in controlling the evolution of the relevant OSS-project. But if one does not belong to the core developers, then the only way to to have a bearing on the project is to submit improvements that are good enough to be accepted. Hence, (iv) is not about adding features that are add-ons, but gettings things implemented e.g. in the Linux Kernel. This leads to a topic, still to be examined in an analytical model: The economics of contributing, as a method for indirect control over the future evolvement of a project. It would be interesting to model this, as with more and more firms getting involved in the 'OSS-game', this indirect control issue will become more important for the further development of OSS.

To sum up: OSS licenses are designed in order to *create* (or: enable), *internalize* and *control* the cumulative effects. Of course, this does not work as nice as exclusively owned networks where the network effects are *created* simply by plugging in, *controlled* simply because there is an exclusive owner who has control over the (further development of the) network, and the network effects can be *internalized* by the price mechanism. In case of OSS, the positive cumulative effects are *internalized* by lowering (potential) development costs, as any cumulative activity yields a new piece of source code (new feature, a fixed bug,...) available for everybody. Of course—as there is no such thing as a free lunch—there is a price to be paid for this, i.e. one has to forgo exclusive rights, exclusive ownership respectively, as this is the necessary condition to *create* the positive feedback effect. However, since today, economic theory has paid surprisingly little attention to the question, how the cumulative effects (and the OSS projects) are *controlled*.

6 Outlook

Still a lot of work has to be done. The arguments of section 4 and 5 have to be presented in a more compact and formal way. Especially the interplay of passive control rights and self-interest of *contributing actively* because of the need for (indirect) control of the project has to be modeled in a formal way. Of course, Y_i^{tc} plays a role in this context. As Y_i^{tc} also determines, whether OSS or CSS is the more attractive strategy, this is close to the modeling of the individual OSS vs. CSS firm strategy decision. Each institutional arrangement has its assets and drawbacks from a firm's perspective. The value of a strategy depends on the firm's own ressources and relative position in the market as well as on the traits and conducts of the other players (customers and other firms). As each strategy corresponds to an institutional arrangement, and as each firm's decision is related to the other firms' decisions, there is a dynamical feedback in chosen institutional arrangements. Because of such interdependencies and feedback-effects, one can expect to derive some intersting dynamical results, and the approach offers the opportunity to model endogenous institutional change, in its broadest sense, regarding the software industry.

References

Aoki, M. (2001), *Toward A Comparative Institutional Analysis*, MIT Press.

Aoki, M. (2007), 'Endogenizing institutions and institutional changes', *Journal of Institutional Economics* **3**(01), 1–31.

Baake, P. & Wichmann, T. (2004), Open source software, competition and potential entry., Berlecon Research Papers 5, Berlecon Research, Berlin.

Barzel, Y. (1997), *Economic Analysis Of Property Rights*, Cambridge Univ. Press, Cambridge [u.a.].

Baumol, W. J. (1977), 'On the proper cost tests for natural monopoly in a multiproduct industry', *American Economic Review* **67**(5), 809–22.

Benkler, Y. (2002), 'Coase's penguin, or, linux and the nature of firm', *Yale Law Journal* **112**(3), 369–437.

Besen, S. M. & Raskind, L. J. (1991), 'An introduction to the law and economics of intellectual property', *Journal Of Economic Perspectives* (1), 3–27.

Bessen, J. E. & Hunt, R. M. (2004*a*), An empirical look at software patents, Working papers, Federal Reserve Bank of Philadelphia.

Bessen, J. E. & Hunt, R. M. (2004*b*), The software patent experiment, *in* OECD, ed., 'Patents, Innovation And Economic Performance', OECD, Paris.

Blind, K., Edler, J. & Friedewald, M. (2005), *Software Patents – Economic Impacts And Policy Implications*, Elgar, Cheltenham [u.a.].

Böhnlein, I. (2003), Anwendung von Aspekten der Neuen Institutionenökonomik auf Open Source Software. Produktion, Verfügungsrechte und Transaktionskosten – eine theoretische und empirische Untersuchung, Master's thesis, Johann Wolfgang Goethe-Universität, Frankfurt am Main.

Brand, A. & Schmid, A. (2005), Koordination in einem Open Source-Projekt, Technical report.

Brousseau, E. (2004), Property rights in the digital space, *in* E. Colombatto, ed., 'Companion To Economics Of Property Rights', Edward Elgar, pp. 438–472.

Brügge, B., Harhoff, D., Picot, A., Creighton, O., Fiedler, M. & Henkel, J. (2004), *Open-Source-Software – Eine Ökonomische und Technische Analyse*, Springer, Berlin [u.a.].

Canfield, K. (2006), 'The disclosure of source code in software patents: Should software patents be open source?', *Columbia Science And Technology Law Review* (6).

Coase, R. H. (1960), 'The problem of social cost', *Journal of Law Economics* **3**, 1–44.

Cowan, R. & Harison, E. (2001), Protecting the digital endeavour. prospects for intellectual property rights in the information society, Research Memoranda 28, MERIT, Maastricht Economic Research Institute on Innovation and Technology, Maastricht.

Crowston, K., Wei, K., Li, Q. & Howison, J. (2006), Core and periphery in free/libre and open source software team communications, System Sciences: HICCS (Hawaii International Conference) Proceedings, pp. 118a–118a.

Dahlander, L. & Magnusson, M. G. (2006), Business models and community relationships of open source software firms, *in* J. B. Schröder & P. J. H., eds, 'The Economics Of Open Source Software Development', Elsevier, pp. 111–130.

Davis, L. E. & North, D. C. (1971), *Institutional Change and American Economic Growth*, University Press.

Demsetz, H. (1967), 'Towards a theory of property rights', *American Economic Review* (2), 347–359.

Dosi, G. (1996), The contribution of economic theory to the understanding of a knowledge based economy, *in* OECD, ed., 'Employment And Growth in The Knowledge-Based Economy', Paris, pp. 81–92.

Eggertsson, T. (1990), *Economic Behavior And Institutions*, Cambridge University Press.

Engel, C. (2002), 'Windows as an institution organizing the markets for applications software', *Journal Of Institutional And Theoretical Economics* (158), 155–162.

Feller, J. & Fitzgerald, B. (2002), *Understanding Open Source Software Development*, Addison-Wesley.

Franck, E. & Jungwirth, C. (2001), Open versus Closed Source. Eine organisationsökonomische Betrachtung zum Wettbewerb der Betriebssysteme Windows und Linux, Technical report.
URL: *www.isu.unizh.ch/fuehrung/Dokumente/WorkingPaper/4full.pdf*

Franck, E. & Jungwirth, C. (2002), 'Das Open-Source-Phänomen jenseits des Gift-Society-Mythos', *WiSt - Wirtschaftswissenschaftliches Studium* **31**(3), 124–129.

Furubotn, E. G. & Richter, R. (2005), *Institutions And Economic Theory : The Contribution Of The New Institutional Economics*, Univ. Of Michigan Press.

Gallini, N. T. (2002), 'The economics of patents: Lessons from recent u.s. patent reform', *Journal Of Economic Perspectives* **16**(2), 131–154. available at http://ideas.repec.org/a/aea/jecper/v16y2002i2p131-154.html.

Gandal, N. (1994), 'Hedonic price indexes for spreadsheets and an empirical test of the network externalities hypothesis', *RAND Journal Of Economics* (1), 160–170.

Garzarelli, G. (2003), Open source software and the economics of organization, Industrial Organization 0304003, EconWPA.

Gehring, R. A. (2006), 'The institutionalization of open source', *Poiesis & Praxis: International Journal Of Technology Assessment And Ethics Of Science* **4**(1), 54–73.

Graham, S. & Somaya, D. (2004), The use of patents, copyrights and trademarks in software: Evidence from litigation, *in* OECD, ed., 'Patents, Innovation And Economic Performance', OECD, Paris.

Gröhn, A. (1999), *Netzwerkeffekte und Wettbewerbspolitik. Eine Ökonomische Analyse Des Softwaremarktes*, Mohr Siebeck, Tübingen.

Hart, O. & Moore, J. (1990), 'Property rights and the nature of the firm', *Journal Of Political Economy* **98**(6), 1119–1158.

Hawkins, R. E. (2004), 'The economics of open source software for a competitive firm', *Netnomics* **6**(2), 103–117.

Heller, M. (1998), 'The tragedy of the anticommons: Property in the transition from marx to markets', *Harvard Law Review* (3), 621–688.

Henkel, J. (2006), 'Selective revealing in open innovation processes: The case of embedded linux', *Research Policy* **35**(7), 953–969.

Jensen, C. & Scacchi, W. (2007), Role migration and advancement processes in ossd projects, International Conference On Software Engineering, To Appear (29), Minneapolis, MN, USA.

Johnson, J. P. (2001), 'Economics of open source software.', Paper.

Kant, I. (1798), *Essays And Treatises On Moral, Political And Various Philosophical Subjects*, chapter 'Of the Injustice of Counterfeiting Books'.

Katz, M. L. & Shapiro, C. (1994), 'Systems competition and network effects. (symposia network externalities)', *Journal Of Economic Perspectives,* **8**(2), 93–115.

Kooths, S., Langenfurth, M. & Kalwey, N. (2003), *Open-Source Software: An Economic Assessment*, Vol. 4 of *MICE Economic Research Studies*, Muenster Institute For Computational Economics, Münster.

Langlois, R. N. (2002), 'Modularity in technology and organization', *Journal Of Economic Behavior & Organization* **49**(1), 19–37.

Leiteritz, R. (2004), Open-Source-Geschäftsmodelle, *in* R. A. Gehring & B. Lutterbeck, eds, 'Open Source Jahrbuch 2004', Lehmanns Media, Berlin, pp. 139–169.

Lerner, J., Pathak, P. A. & Tirole, J. (2006), 'The dynamics of open-source contributors', *The American Economic Review* **96**(2), 114–118.

Lerner, J. & Tirole, J. (2002), 'Some simple economics on open source', *Journal Of Industrial Economics* **50**(2), 197–234.

Lessig, L. (1999), *Code And Other Laws Of Cyberspace*, Basic Books, New York, NY.

Lessig, L. (2006), *Code: Version 2.0*, Basic Books.

Libecap, G. (2004), The effect of transaction costs in the definition and exchange of property rights: Two cases from the american experience, *in* E. Colombatto, ed., 'Companion To Economics Of Property Rights', Edward Elgar, pp. 438–472.

Libecap, G. D. (1989), *Contracting for Property Rights*, Cambridge University Press.

Liebowitz, S. J. & Margolis, S. E. (1994), 'Network externality: An uncommon tragedy.', *Journal Of Economic Perspectives* **8**(2), 133–150.

Liebowitz, S. J. & Margolis, S. E. (2002), Network effects, *in* M. Cave, S. Majumdar & I. Vogelsang, eds, 'Handbook Of Telecommunications Economics', Vol. 1, pp. 76–94.

Locke, J. (1698), *Two Treatises On Government*.

McGowan, D. (2001), 'The legal implications of open source software', *Illinois Law Review* (1), 241–304.

Menell, P. S. (2000), Intellectual property: General theories, *in* B. Bouckaert & G. De Geest, eds, 'Encyclopedia Of Law And Economics, Volume II. Civil Law And Economics', Edward Elgar, Cheltenham.

Mundhenke, J. (2004), 'Chancen und Risiken von Softwarepatenten', *Die Weltwirtschaft* (4), 417–438.

Mustonen, M. (2005), 'When does a firm support substitute open source programming?', *Journal Of Economics & Management Strategy* (1), 121–139.

Mustonen, M., ed. (2001), *Copyleft - The Economics Of Linux And Other Open Source Software.*, University Of Helsinki.

Nüttgens, M. & Tesei, E. (2000), Open Source: Marktmodelle und Netzwerke, Veröffentlichungen des Instituts für Wirtschaftsinformatik, Saarbrücken.

Osterloh, M., Rota, S. & von Wartburg, M. (2001), Open source - new rules in software development, Technical report.

Pasche, M. & von Engelhardt, S. (2004), Volkswirtschaftliche Aspekte der Open-Source-Softwareentwicklung, Jenaer schriften zur wirtschaftswissenschaft.

Picot, A., Dietl, H. & Franck, E. (2005), *Organisation: eine ökonomische Perspektive*, 4., überarb. und erw. aufl edn, Schäffer-Poeschel.

Powell, W. (1996), Weder Markt noch Hierarchie: Netzwerkartige Organisations-Formen, *in* P. Kenis & V. Schneider, eds, 'Organisation Und Netzwerk – Institutionelle Steuerung in Wirtschaft Und Politik', Campus-Verl., Frankfurt/Main [u.a.].

Quah, D. (2003), Digital goods and the new economy, CEP Discussion Papers 563, London School of Economics, London.

Romberg, T. (2003), Herstellerübergreifende Wiederverwendung von Komponenten, *in* 'Handbuch Zur Komponentenbasierten Softwareentwicklung', Frauenhofer-Institut Für Experimentelles Software Engineering / Forschungszentrum Informatik.

Rossi, C. & Bonaccorsi, A. (2006), Intrinsic motivations and profit-oriented firms in open source software: Do firms practise what they preach?, *in* J. B. Schröder & P. J. H., eds, 'The Economics Of Open Source Software Development', Elsevier, pp. 84–109.

Rossi, M. A. (2006), Decoding the free/open source software puzzle: A survey of theoretical and empirical contributions, *in* J. Bitzer & P. Schröder, eds, 'The Economics Of Open Source Software Development', Elsevier, pp. 15–55.

Schiff, A. (2002), 'The economics of open source software. a survey of the early literature', *The Review Of Network Economics* **1**(1), 66 – 74.

Shy, O. (2001), *The Economics Of Network Industries.*, Cambridge University Press, Cambridge.

von Engelhardt, S. (2006), Die ökonomischen Eigenschaften von Software, Jenaer Schriften zur Wirtschaftswissenschaft, *A translated version will be published soon.*

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

Weber, S. (2004), *The Success Of Open Source*, Harvard University Press.

Wendel de Joode, R. V., Bruijn, J. A. d. d. & Eeten, M. J. G. V. (2003), *Protecting The Virtual Commons – Self-Organizing Open Source And Free Software Communities And Innovative Intellectual Property Regimes*, T.M.C. Asser Press, The Hague.

Westarp, F. v. (2003), *Modeling Software Markets*, Physica-Verlag, Heidelberg [u.a.].

Wey, C. (1999), *Marktorganisation durch Standardisierung*, Ed. Sigma, Berlin.

White, A. G., Abel, J. R., Berndt, E. R. & Monroe, C. W. (2004), Hedonic price indexes for personal computer operating systems and productivity suites, NBER Working Papers 10427, National Bureau of Economic Research, Inc.