Westfälische Wilhelms-Universität Münster

MICE Economic Research Studies

Vol. 4

Open-Source Software

An Economic Assessment

Stefan Kooths, Ph. D.

Markus Langenfurth, Ph. D.

Nadine Kalwey, *Diplom-Volkswirtin*

DRAFT TRANSLATION

Muenster

December 2003

mice.uni-muenster.de

MICE

Muenster Institute for
Computational Economics

University of Muenster

Editors:

Prof. Gustav Dieckheuer, Ph. D. / Stefan Kooths, Ph. D.

Kooths, Stefan / Langenfurth, Markus / Kalwey, Nadine:
Open-Source Software – An Economic Assessment
http://mice.uni-muenster.de/mers/mers4-OpenSource_en.pdf

MICE Economic Research Studies

ISSN 1612-9032

Westfälische Wilhelms-Universität Münster

Muenster Institute for Computational Economics

Fliednerstrasse 21

D-48149 Muenster/Germany

http://mice.uni-muenster.de

Management Summary

Open-Source Software: An Economic Assessment

1. No Market at the Core – Open-Source Development

Open-source software is deliberately developed outside of market mechanisms, as the main purpose of making the source code freely available is to prevent a price-controlled software market from evolving in the first place. This is a fact that commercial open-source business models cannot alter, especially not in the packaged software sector. The business transactions of such models take place in complementary markets and have at most an indirect effect on open-source software development. However, in any economy based on the division of labor, the market fulfills important information, coordination and incentive functions: it creates an equilibrium between customer wants and product supply (customer sovereignty), steers scarce resources towards their best-possible use (optimum allocation of resources), generates income and distributes it as warranted by performance (productivity-oriented factor compensation) and provides innovation incentives (progressive function). In such a market, prices are the main information medium for suppliers and demanders; without prices, the markets are unable to fulfill the above functions. When software is distributed free of consideration ("free"), however, it lacks this key price component, which results in the substantial economic and functional deficits of the open-source model.

2. "Happy Engineering" - Developer Orientation Is Not Customer Orientation

Open-source developers are involved in projects that fit in well with their personal preferences: they derive most of their motivation from an individual interest in solving a problem, the excitement of a technological challenge or the hope of building a reputation for themselves. Thus, the interests of developers greatly determine the type and scope of the software supply. Because of this incentive mechanism, developers produce mostly sophisticated solutions for advanced users. However, software supply should not be determined by what is technically possible but by what the user actually wants and is able to use and pay for. It is the customer who, as a sovereign in the market process, ultimately determines supply with his or her product decision. Suppliers of proprietary software can only survive in an efficient market if their product supply matches the customers' wants. Their market-research investments serve to identify customer wants and, by extension, to contribute to their own survival in the market process. The pricing of their products (based on realizable selling prices and volume (sales) on the one hand and on the calculation of the resources utilized in the development of the product on the basis of factor prices and volume (costs) on the other) generates profit or loss signals that software suppliers can use as guides in an efficient market. If, however, there is no market, then there is no reliable mechanism to steer the interests of developers towards the actual wants of customers, either. Customer sovereignty cannot be accomplished without product prices – software users who do not write programs turn into passive recipients of what the open-source developer community puts out.

3. Not for Nothing, but Sometimes for Naught - Regulating Open-Source Resources

Economizing means utilizing available, scarce resources in a way that, in the customers' estimation, uses these resources in the best-possible fashion. If software is available free of consideration, then the work invested in its development cannot be directly compensated for monetarily. This lack of pricing transfers directly to the upstream factor markets where it nullifies the balancing mechanisms of the market economy. For this reason, many developers may be working on programs that nobody wants, or the programs that users want may not be developed. Even a wide distribution of an open-source product is not a reliable indicator of a successful utilization of resources as it does not offer any clues as to whether the use of the development capacity for an alternative product would have led to a higher satisfaction of customer needs (nontransparency of opportunity costs). The ability to copy software basically free of charge has been interpreted in the open-source discussion to mean that there is no rivalry among users concerning software products, from which a public-good attribute and a partial market failure are then derived. This view, however, is too short-sighted, as it only refers to existing (regardless of how it was produced) software (lack of ex post rivalry). On other hand, before new software can even be developed, there must be competition for the use of scant development capacity (ex ante rivalry). This essentially determines which products should or should not be developed. It is especially in the software sector with its highly qualified workforce that resources should be utilized according to market-economy, i.e., productivity-oriented criteria so as not to waste the economically scarce resource of "development capacity".

4. Second-Best Solution – Commercial Open-Source Business Models

As the commercial exploitation of open-source software itself is restricted, commercial business models use open source to promote the sale of complementary services and products. In so doing, commercial business models are dependent on being able to market the complementary product as exclusively as possible. Their incentives for investing in open-source software are not directed by the open-source product but indirectly by the value of the open-source software to their marketable product. Commercial business models must indirectly finance all their investments in open-source development with the profits from complementary products, provided they are not customizing software for specific clients. This form of cross-subsidization distorts the price structure of the market economy, which may lead to sizeable distortions in the production structure. For example, financing the development of software from service activities would drive up service prices beyond the actual costs incurred in development, leading to a lower demand for these services than would be possible in a free-enterprise system. Varying economic activities (software development, services) should also be priced independently of one another in order to clearly signal to the demander which resources are actually used in the process of satisfying the demander's wants.

5. Weak Proprietary Software Does Not Mean Strong Open-Source Software

The open-source model does not represent a basic alternative form of software development. One cannot strengthen open source by weakening the proprietary software market, as the development of open-source software is contingent upon there being a strong proprietary software market. The proprietary market serves open-source development as a wellspring of resources for jobs, income and innovative product ideas.

6. Open-Source Software Does Not Aid SMEs in the IT Sector

Far from offering new business opportunities, open-source software offers only some of the opportunities already available in the proprietary software market. Promoting open source is not a suitable locational means of supporting SMEs in the IT sector. Except as regards custom software work, the nonmarket coordination mechanism fails to contribute to the creation of value in development, as opposed to the proprietary software market. If software is available free of consideration, its development – unlike in the proprietary software market – does not generate profit, income, jobs or taxes. Within the scope of the Microsoft partner model alone, independent software companies with a total of 45,000 employees produce Microsoft-related software products worth 6.6 billion euros. Business models based on open source that offer complementary services and products are not fundamentally different from proprietary offerings in this complementary field. However, the cross-subsidization of the nonmarket core of open-source software causes value-added to be lower than in the production of proprietary software. Consequently, value-added in the open-source model is lower overall.

7. Promotion of Open Source – Not a Competition-Policy Tool

The position of individual suppliers in certain market segments of the IT sector (in particular in desktop operating systems and office software) does not constitute a justification for promoting open-source software. State support/subsidization of competitors in highly concentrated markets is not a competition-policy tool because such interventions require a level of information that government authorities simply cannot have (e.g., future market trends, optimum market structure). As an IT demander, the state should therefore – as stipulated in budgetary laws – be guided strictly by economic considerations (TCO analyses), deliberately abstain from influencing market developments and leave it up to anti-trust commissions to enforce the rules of competition. Even if one disregards these basic regulatory principles for any reason whatsoever, there is still one question that remains: why should a production process that suffers from the above coordination deficits be supported at all?

Table of Contents

List of Illustrations and Tables

Introduction

Since the dreams of the internet boom and the associated ideas of a New Economy have not materialized, hopes are now revolving around a new buzzword in the world of IT: open-source software. Once again, there are claims that a new and differently regulated economy will emerge, this time based on software whose primary economic attributes are the nonexistence of property rights and – consequently – its availability free of charge. The exchange of "goods for money" is to be superseded by an exchange of "gifts for reputation" and participation in a developer community based on reciprocity. The declared goal is to make software generally available without licensing fees, thereby nullifying the market-coordination mechanism that is controlled by the price system. The expectations of the public associated with open-source software are manifold: more transparency, more democracy, more jobs particularly in small and medium-size enterprises, and last but not least, a contribution to consolidating public finances with allegedly free software.

Advocates of producing software that, at the core, dispenses with prices and the market process, are often fundamentally skeptical of market coordination mechanisms. The open-source development of packaged software deliberately shuns the market and thus seems to constitute an engineering method that does not conform to the processes characteristic of a market economy. This absence of market processes manifests itself as a nonexistence of prices and the free distribution of software. The open-source licenses on the software aim to suppress any ownership claims to the software and prevent prices from being established for it. In the end, the developed software cannot be used to generate profit.

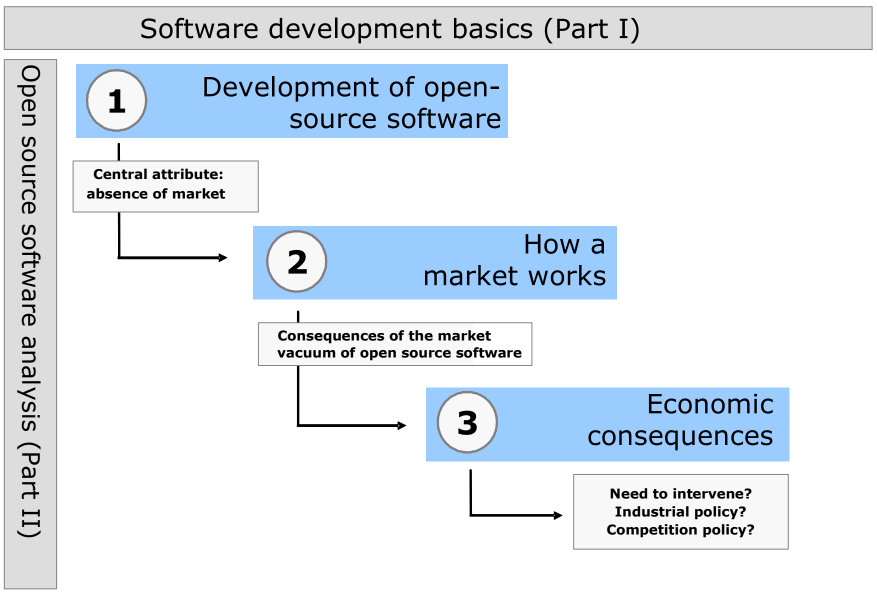
Be that as it may, business models based on open-source software do exist. These business models must, however, finance basically all their software-development investments from complementary services. There is one exception: custom contract software. As this software is tailored to the special interests of each customer, it is not mass-marketed, which makes the (complementary) service of developing the software tantamount to the sale of a one-time license. And while the software is subsequently made available for free public use, there is generally no demand for it: it contains special customized features and will therefore not be deployed by any users other than the customer. In sum, the advantages attributed to open-source software cannot be realized in custom software.

However, any complementary business models that are not based on custom contract development are severely limited. The returns from investments in a particular open-source development are simultaneously available to all other suppliers, who can then benefit from a competitor's investments in development and offer their own complementary services without having to finance investments on their own. This in turn reduces the overall willingness to invest in software development. Unlike custom software development, the domain of packaged software does not constitute an economically sustainable foundation for open-source business models. Despite the fact that there are business models that exploit open-source software, software development, as the core of the open-source model, remains nonmarket in principle.

In a market economy, however, prices and the market play a vital role as a coordination tool: the suppliers' production plans are matched to customer demand, scarce resources are steered towards their most productive use, and innovation incentives are provided in the form of property rights in the finished product. In such an economy, prices assume a central regulatory function. They indicate relative shortages and help suppliers determine what value demanders place on a given good and which goods are in highest demand. Only prices render an economy based on the division of labor possible; without them, there can be neither sales nor income. No other measure even comes close to providing an equally effective assessment of customer wants, production facilities and new market opportunities. If open-source developers choose not to price their software products, the market is effectively stripped of its central regulatory functions. For that reason, the quality of not bearing a price at all means a great deal more, economically speaking, than the quality of being free in the eyes of demanders.

This study examines the extent to which the open-source model constitutes an alternative to the production of proprietary software. It primarily examines how the absence of market processes impact this form of software production.

Illustration 1: Design of the Study


Software development basics (Part I)
Open source software analysis (Part II)
1
Development of open-source software
Central attribute: absence of market
2
How a market works
Consequences of the market vacuum of open source software
3
Economic consequences
Need to intervene? Industrial policy? Competition policy?

The study is split up into two sections. The first section describes the basics of the software market and software development. The second part briefly describes the beginnings of open-source software, followed by an economic assessment of open-source software and a discussion of its economic consequences.

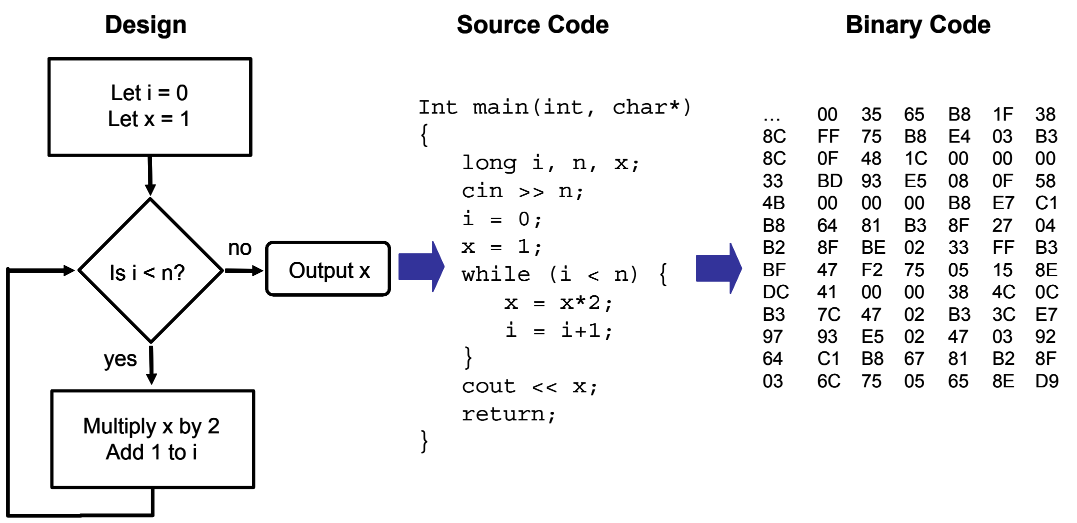
1. Basics of the Software Market

1.1 Attributes of Software

1.1.1 Steps of Software Development

Simply speaking, software consists of a list of data-processing instructions. Software is an intellectual product, comprising programs, procedures, guidelines and documentation required to give instructions to a computer and to have tasks performed. It contains, in their entirety, the formalized program statements and functions required to run a computer (operating system, programming languages and their compilers or interpreters, programs).[1]

Illustration 2: Steps in Software Production



```
Int main(int, char*)
{
    long i, n, x;
    cin >> n;
    i = 0;
    x = 1;
    while (i < n) {
        x = x*2;
        i = i+1;
    }
    cout << x;
    return;
}
```

```
...  00  35  65  B8  1F  38
8C   FF  75  B8  E4  03  B3
8C   0F  48  1C  00  00  00
33   BD  93  E5  08  0F  58
4B   00  00  00  B8  E7  C1
B8   64  81  B3  8F  27  04
B2   8F  BE  02  33  FF  B3
BF   47  F2  75  05  15  8E
DC   41  00  00  38  4C  0C
B3   7C  47  02  B3  3C  E7
97   93  E5  02  47  03  92
64   C1  B8  67  81  B2  8F
03   6C  75  05  65  8E  D9
```

Source: EVANS AND REDDY (2002), p. 5.

The ideal development of software typically takes place in three steps. First, the design is drafted as a blueprint that reflects the software's purpose. Next, the actual programming takes place: the programmer writes, on the basis of the blueprint, the instructions in a particular programming language. These instructions are called the "source code" of the software. The source code is the instruction sequence of software and is what determines its functionalities.

Higher programming languages (such as PASCAL, C or C++) are based on written language, and on written English in particular. For example, numerous

[1] Cf. JANKO, BERNROIDER AND EBNER (2000), p. 13.

programming languages contain commands that use the words "if" or "when". The last step in software development is the final translation into binary code. In this step, a compiler or interpreter translates the source code into a form that can be used by the computer; the source code becomes the object code or binary code (consisting of zeros and ones or on/off instructions for the processor circuits). Illustration 2 shows the individual steps of software development.[2]

The following trends can be identified within the domain of software development:[3]

- An increasing importance of software products over hardware products
- An increasing importance of software-related services over software products
- An increasing complexity of software projects
- Rising quality standards
- A trend from custom software to packaged software
- An increasing number of new versions based on existing software.

The less these individual steps in the development of software are open to the public, the more effectively protected is the knowledge that has gone into the development of the software, and the more difficult it is to program one's own software based on this knowledge. If only the software design is available as a blueprint, then, even though the structure and logic of a software product are known, the programming itself still has to be done independently. If the source code can be accessed, the sequence and the individual programming steps can be traced. It is also possible to modify the software or copy individual components. If only the binary code, incomprehensible for humans, is available, then conclusions can no longer be drawn concerning the structure and the programming of the software.[4]

In order to protect the author's intellectual property in the program, commercial software is often distributed in binary form. While a copyright or patent may legally protect intellectual property rights, it is not always possible to enforce them.[5] On

2 Cf. EVANS AND REDDY (2002), p. 5. The binary code in Illustration 2 is written in hexadecimals – a simple way of writing zeros and ones.

3 Cf. BALZERT (1996), p. 27 and BERLECON RESEARCH (2002c), pp. 24-25.

4 Cf. BERLECON RESEARCH (2002c), p. 11, GRASSMUCK (2002), pp.. 233-234 and GRÖHN (1999), p. 4, so quoted.

5 Cf. SCHMIDT AND SCHNITZER (2003), p. 4.

the other hand, translating the binary code back into the source code - what is known as reverse engineering – requires a Herculean effort. Individual software components or the software as a whole can also be protected by patents or copyright.[6] This bans unauthorized agents from copying the software or - provided that the source code is accessible - from incorporating individual software components into other programs without the author's permission.[7]

1.1.2 History of the Market Structure

The idea of assigning certain tasks to general-purpose computers through stored-program control was first conceived some 50 years ago. Since then, the software industry has gradually evolved from a vertically integrated to a horizontal, nonintegrated market structure.[8]

Until the late 50's, the software market consisted of custom software projects in which the U.S. government and other clients awarded contracts to hardware firms or independent software companies. As mainframes became more widely used, software demand gradually outstripped the development resources of corporate computer departments and hardware manufacturers. Increasingly, this demand was addressed by independent software producers. This led to the launch, in the 60's, of the first mass-marketable software packages. However, these products were still tailored to the special needs of large companies.

The period stretching from the 70's to the early 80's experienced an uncoupling of hardware production and software engineering. IBM decided to market hardware and software separately ("unbundling" policy), resulting in the emergence of a wide range of standard business software products for deployment in various industries. This is also the era that witnessed the birth of business software providers such as SAP (1972) and BaaN (1978).

Until the early 80's, virtually all computers were mainframes and most software was developed for mainframes. This continued until IBM unveiled its mass-

[6] For example, the algorithm for creating MP3 files has been patented by Fraunhofer Gesellschaft. Every software product that uses this algorithm to create MP3 files must be licensed by Fraunhofer Gesellschaft.

[7] Cf. Evans and Reddy (2002), p. 6.

[8] Cf. Hoch, et al. (2000) pp. 259 et. seq. and Janko, Bernroider and Ebner (2000), pp. 16-17 for more on software history. See also www.softwarehistory.com.

marketable Personal Computer on August 12, 1981.[9] IBM was still marketing its hardware and software separately at this point, encouraging the creation of an independent market for end-user software.[10]

Not only did the software sector break away from the hardware sector, but an independent service sector developed. IT service providers offered consulting, implementation, training or data-center services as well as custom software engineering. However, the line between software developers and software service providers is sometimes artificial and often hard to draw. For example, many business software developers also have service departments. Likewise, IT services may frequently contain development components.[11]

These developments fundamentally shaped the current structure of IT market. The following section examines the economic characteristics of the software market.

1.2 An Economic Analysis of the Software Market

1.2.1 Economic Attributes of Software

An examination of software based on economic criteria reveals special attributes. These software attributes impact both the development process and competition in the software market. This section presents the economic characteristics of software from the supply and the demand perspectives.[12]

1.2.1.1 Supply Side Attributes

- **Extremely high development costs ("first copy costs")**

  Software is a product of intellectual property rights. As such, most production costs are incurred as sunk costs during the development and pre-launch testing.[13] This may create a need to commercially exploit newly developed software as quickly as possible.

---

9 Personal computers had existed before: the Altair 8800 in 1975, and the Apple II in 1977. However, they were not able to establish themselves as longer-lived standard platforms.

10 Cf. LEHRER (2000), p. 590.

11 Cf. LEHRER (2000), p. 589.

12 See also BALZERT (1996) for more on attributes.

13 Cf. OECD (2002), p. 105.

- **Extremely low marginal cost of production**

  In the case of packaged software, the finished product is available in digital form and can be copied any number of times. The copy costs and hence the marginal costs of production are extremely low, which is why the software industry is characterized by degressive average cost trends.[14] As a result, there are hardly any output restrictions on the production of software. If software is distributed physically, the only supply-side bottlenecks that can occur are in copying, packaging and distribution.

- **Economies of scope in production**

  Once written, program elements can be used in other programs. For this reason, economies of scope play a vital role as well. Interfaces may also be defined that allow other producers to also benefit from economies of scope by providing them with individual software elements or components.[15] For example, the so-called APIs (application programming interfaces) within the Windows operating system make it possible to call individual components of the operating system.

- **Network effects on the supply side**

  The more applications there are for a certain operating system, the more attractive this system becomes for the user. The growing number of users in turn makes it more attractive for developers to develop new software for a certain platform. Unlike returns to scale in the production of software, network effects are not confined to any one company. Other suppliers can produce for the same network as well.[16] In this case, they capitalize on an established standard as there is no need to design their software for different platforms.

- **Intangibility**

  It is difficult to measure the value of software because it is intangible and consists to a great extent of know-how. It is also difficult to quantify, in monetary terms, the progress made in software development. This in turn

[14] Cf. OECD (2002), p. 105 and GRÖHN (1999), p. 5.

[15] Cf. GRÖHN (1999), p. 5.

[16] Cf. GRÖHN (1999), p. 3.

makes it hard to establish product attributes like quality or to compare products.[17]

- **Internationality**

  Because of the attributes of software, its production and distribution are inherently internationalized. Both team development and product distribution can be dispersed across the entire globe. As marginal costs of production are low, it is only a small step to distribute the developed software product worldwide once the menu language has been changed. Internet distribution allows even small software suppliers to enter the global market in its entire breadth at once.

1.2.1.2 Demand Side Attributes

- **No wear and tear, no expiration date - but: software becomes obsolete**

  Software does not wear down, nor does it come with an expiration date. So theoretically, it can be used forever. Software functionality is limited only by the performance of the hardware on which it is installed. As hardware, and processor technology in particular, is enhanced, it provides an opportunity to enhance software and extend its functionality or area of application. If users wish to exhaust all the capabilities offered by a new hardware generation, they will have to update their software. Software becomes obsolete whenever a new version contains new, useful functions or when that new version performs existing functions better.

- **Network effects on the demand side**

  The more users deploy a certain operating system or application software, the easier it is to exchange files and get support in using the program. Consequently, the utility derived from the use of a given program increases with the number of other users that also use that program. By opting for a certain operating system or software, the user joins the network of people using that software.

  However, the boundaries between different networks need not be unbridgeable. For example, formats can be converted to ensure data readability on different platforms and thus enable users to change to, or work together with, other networks.

[17] Cf. JANKO, BERNROIDER AND EBNER (2000), p. 14.

- **Experience good and lock-in effects**

  Software is an experience good. As a rule, the quality and suitability of a certain software product are only revealed after considerable use and a long period of familiarization. This, along with the network effects, may result in a lock-in effect. Once users have familiarized themselves with a certain software product and ensured that all the files are available in a format accepted by that software, they will be reluctant to move to another software product. This move will only make sense for them if the added utility exceeds the necessary costs of learning how to use the new software.

- **Nonrivalry in consumption**

  A given software copy can be physically installed as often as one likes and can be used by different users. There is nonrivalry in consumption.

It then appears that the most important attributes are network effects, nonrivalry in consumption and excludability options in the software market. For that reason, these aspects will be dealt with in more detail in the following two sections.

1.2.2 Network Effects

Network effects are a special case of technological external effects. External effects occur when someone is affected by consumption or production activities of households or companies without paying or being compensated for these activities.[18] When external effects occur, market prices are distorted and do not reflect the actual shortage and utility circumstances as the market does not compensate for all of the utility or loss.[19] The amount of external effects is defined as the difference between the social costs and utility and the private costs and utility compensated for by the market. If the state does not intervene, goods generating a positive added utility are undersupplied, while goods with negative external effects are oversupplied.

The network externalities occurring in the software market are a special case of external effects; they are characterized by the shared use of the same good. Network externalities occur when a user's being connected to a network (here the use of the same software) changes the utility enjoyed by other users in the network

[18] Cf. FRITSCH, WEIN AND EWERS (2003), pp. 92 et seq. GROSSEKETTLER (1995), p. 510.

[19] "External" thus means "located outside the price system as the main coordination mechanism of a market economy". GROSSEKETTLER (1995), p. 510.

without these changes being included in the market price.[20] In technical networks such as a telecommunications network, network externalities can arise when network subscribers are physically connected, or they can come about in virtual networks that are connected through a uniform standard (e.g., a shared PC operating system).[21]

There is a difference between direct and indirect network effects. The direct effect is that the use of a program is advantageous just because many other users are using that program as well. The indirect effect lies in the availability of complementary products and services. The value of a software program therefore depends on the availability of complementary products and services.

If there is a positive correlation between the utility derived from a network and the number of participants in it, then network externalities will become economically relevant upon reaching a *critical mass* of network subscribers.[22] Setting up a network or switching to an alternate network is worthwhile only if there is a sufficient number of participants. If this critical number of participants is not reached, the network may not develop or users may remain in the current network. Such a lock-in effect may act as an entry barrier for alternative providers who wish to set up a network that is technologically identical to that of the established supplier, or it may cause users to remain in a technologically obsolete network. In addition, a network may be fragmented if several similar networks use incompatible technologies and thereby prevent network externalities from being fully utilized. Thus, setting up a second network does not make good economic sense.

The problem of the lock-in effect appears especially in those cases when switching networks entails irreversible costs and when there is incomplete information and asymmetrical preferences. Network switchers with a strong preference for the new technology cannot be sure that they will be joined by other users. If they adopt a strategic "wait and see" attitude, the switching process might not even get started.

[20] Cf. WEIZSÄCKER, VON AND KNIEPS (1989), p. 458, KATZ AND SHAPIRO (1994), pp. 93-115.

[21] Cf. BLANKART AND KNIEPS (1992), p. 73. These label tangible networks as hardware networks and intangible networks as software networks. Cf. also KLODT ET AL. (1995), p. 40.

[22] Cf. KLODT ET AL. (1995), p. 40, and BLANKART AND KNIEPS (1992), p. 79.

1.2.3 Nonrivalry and Excludability

The previous section pointed out the significance of excludability and nonrivalry in consumption. This section examines these attributes within the context of the public goods theory.

The public goods theory makes it possible to draw conclusions about the functionality of a market from present attributes of goods. The theory does not focus on whether a market is organized competitively or monopolistically but on identifying functional defects that prevent the development of self-organized markets.[23] If a public good is present, the market cannot carry out its allocation function: either there is no supply at all, or supply and demand are not synchronized, thereby triggering an undersupply or an over-use of the public good.

The main distinguishing feature between public and private goods is the ability of several demanders to use public goods simultaneously. While each consumer articulates his or her demand for private goods individually, the demand for public goods must be organized beforehand in order to determine total need and financing.[24] If demand is not organized accordingly, then a market for this good will not develop. Rules can be derived for the financing and supplying of different categories of public goods. These categories of goods are identified using the criteria of *private excludability* and *rivalry in consumption*.[25] While the degree of exclusion ε shows whether demanders can be excluded from enjoying a good and thus enabling a self-organized supply to develop through the market, the degree of rivalry λ provides information on the cost of including an additional user and, by extension, on the economically reasonable price of a good.

Wherever nonexcludability ($\varepsilon = 0$) exists, individual demanders cannot be excluded from enjoying the good at a reasonable cost.[26] In this case, the ownership rights in a good are not allocated or cannot be enforced. A market will not develop unless it is possible to exclude users from the consumption of a good. If it is not possible to effectively exclude demanders, then a market for the respective good will not develop as potential consumers – expecting to have to contribute to financing themselves - will not be willing to freely articulate their needs. As the

[23] Cf. BURR (1995), p. 41, and GROSSEKETTLER (1991), pp. 119-120.

[24] Cf. GROSSEKETTLER (1995), p. 499.

[25] Cf. GROSSEKETTLER (1995), p. 496.

[26] An exclusion technique is economically defensible if it reduces crowding and waste costs, prevents freeloading and if the value of the avoided costs is not lower than the value of the exclusion costs.

consumer knows that he or she cannot be excluded from consumption, he or she could use the good parasitically without making a financial contribution. This leads to the "prisoners' dilemma", in which every individual acts rationally and yet, in macroeconomic terms, an inefficient market result comes about.[27] A company will not be willing to offer a good with a nonexcludability attribute as the company does not, under private law, have any means to enforce a financial contribution from users. As a result, this product will not be produced.[28]

Table 1: Categories of Goods

	Degree of rivalry	
Excludability	**Nonrivalry** $\lambda = 0$	**Rivalry** $\lambda = 1$
Nonexcludability $\varepsilon = 0$	Prototypical public goods (e.g., lighthouse)	Quasi-public goods (e.g., ozone layer)
Excludability $\varepsilon = 1$	Club goods (e. g., software)	Private goods (e.g., food)

Source: GROSSEKETTLER, H. (1995a), p. 499.

Consequently, the attribute of excludability is of material importance in software engineering. If it is not possible to effectively exclude users or enforce licensing rights, companies have no incentives to invest in the production of software or to distribute it commercially because of the aforementioned freeloader problem. This aspect is especially relevant to open-source software, which expressly rules out exclusion and as such the enforcement of ownership rights in the used licenses.

At a zero degree of rivalry, no additional units of this good are necessary to supply additional demanders, the marginal costs of supply then equal zero, i.e., providing the additional quantity does not incur any costs.[29]

Four groups can be formed by combining the extrema of the criteria for classifying public and private goods (Table 1).[30] A polar or *prototypical public good* is when excludability is not possible and an additional user can be supplied at no additional cost. In the case of *quasi-public goods*, additional users cause a loss of utility as

[27] Cf. BURR (1995), p. 28.

[28] Cf. GROSSEKETTLER (1995a), p. 496.

[29] Cf. GROSSEKETTLER (1995a), pp. 502-504.

[30] Concerning the classification of goods, cf. GROSSEKETTLER (1995a), pp. 500-501.

soon as overcrowding symptoms appear at the capacity limit. In this case, the costs of over-use would be necessary to exclude additional demanders; this, however, is either technically not feasible, or it is economically ineffective. The ownership rights in quasi-public goods are not attributable to individual persons but are assigned to the public or to a certain group.[31] The danger of over-use and hence the destruction of the good is a characteristic problem of quasi-public goods.[32] In the case of *club goods*, on the other hand, authorized and nonauthorized users can easily be segregated by contractual means (in the case of software, with licenses). There is no rivalry among the authorized users, however. For *private goods*, rights of disposal can be assigned, and there is rivalry in consumption.

In this model, software can be categorized as a club good. An exclusion is possible by explicitly defining rights of disposal; there is no rivalry in use. This categorization of software is associated with a recommendation to supply the software privately. There is no need for state intervention.

1.2.4 Competitive Traits of the Software Market

The attributes of software are also the source of the competitive traits of software. These traits are presented below:

- **Competition as an innovation race and competition for the market**

  These software attributes create an innovative dynamic that is specific to the software market. Low marginal costs and high fixed costs for research and development may motivate companies to sell as many copies as possible in the shortest-possible period in order to finance the development costs.

  The network effects that exist on the software market also increase the pressure of having to tap new markets as quickly as possible. The ability to be the first to establish a given technology or a file format makes it easier to corner the entire market. Because of these first mover advantages, competition *for* the market (the network) is likelier than competition *in* the market.

  Once a market has been occupied, it is difficult to develop a me-too-product with marginal additional functionality and to launch it on the market, unless the development costs are very low. However, price competition can quickly destroy the profitability of a me-too-product. For that reason, competitors attempt to

[31] Cf. BURR (1995), p. 30.

[32] Cf. GROSSEKETTLER (1991), p. 70.

differentiate their products, or they avoid markets that have already been occupied.[33]

The described factors also encourage the concentration of the software market.[34] The emergence of a dominant technology or company in the software market is not necessarily the result of market failure but may be attributable to the specific supply and demand conditions in the software market.[35]

- **Competition with one's own products**

  As software is not consumed, a supplier in a saturated market is also constantly challenged to further develop the product in order to survive in the market.[36] Software is only repurchased and replaced if the new version contains noticeable improvements. If a company wants to sell new versions of a software product, then competition also consists of becoming better than the previous version.[37] As such, inferior technology is unable to permanently dominate, as it is squeezed out either by competing products or by new versions of that same software.

- **Two-sided market**

  The operating systems market, as part of the software market, is essentially a two-sided market, which is distinguished by network effects on the supply and demand sides. Typical platforms with the properties of a two-sided market are operating systems, game consoles and even internet portals. The economic value of a platform consists of bringing suppliers and users together. Illustration 3 shows a simplified model of a two-sided market.

  The more suppliers there are offering products or services for a given platform, the more attractive it is to users. At the same time, the more users opt for a certain platform, the more attractive it is to suppliers. Supply on a two-sided market therefore depends on the number of demanders using a given platform. Likewise, the more suppliers provide products or services for a platform, the more demanders will use it.

[33] Cf. EVANS AND REDDY (2002), p. 16.

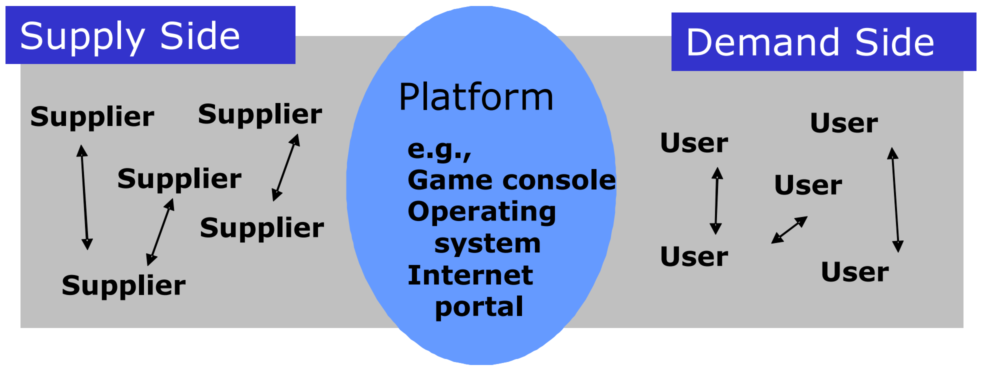
[34] The concentration tendency is stronger in the desktop market than in the market where professional IT employees use software, where training and popularity barriers generally are not quite as high. It is easier for IT employees to learn new programs, and they do not base their purchase decisions on popularity. Cf. HOCH ET AL. (1999).

[35] Cf. EVANS AND REDDY (2002), p. 17 and OECD (2002), p. 105.

[36] Cf. GRÖHN (1999), p. 3.

[37] Cf. SCHMIDT AND SCHNITZER (2003), p. 8.

Illustration 3: Two-Sided Market



The decision to use a certain platform is not necessarily associated with specific investments, as in the case of a game console. For example, users could quite effortlessly switch between different internet pages where goods are auctioned off. In reality, however, suppliers and users tend to congregate around those platforms that are accessed by most of the other suppliers and users.

- **Low entry barriers, constant threat from new suppliers**

As other companies are free to exploit the network effects by producing for the same network as the dominant supplier, markets with network effects are more vulnerable than markets with returns to scale.[38]

The software market's entry barriers for new suppliers are low. Though high development costs are incurred prior to market entry, the additional investments required in order to be able to enter the market are minimal compared to the production of physical goods. Short innovation cycles and technological progress also provide favorable market entry opportunities for new suppliers and threaten the dominance of established companies.[39]

The size of the market entry barriers varies, however, with the significance of the network effects on the supply and demand sides. The more the utility derived from a platform or technology depends on these network effects, the greater the additional utility derived from an alternate platform must be, and the more difficult it is to enter the market.

[38] Cf. GRÖHN (1999), p. 3.

[39] Cf. SCHMIDT AND SCHNITZER (2003), p. 7.

- **Fragile market leaders**

  The comparatively low entry barriers promote high rates of innovation. Numerous new software technologies have been brought to market in the past by small, innovative companies. Thanks to the superiority of their technology or an innovative application, they were able to quickly acquire huge market shares and take the place of established suppliers. As long as they have an innovative product, they do not need any additional investments to enter the market. As such, the success of a new product in the software market does not depend on the size of the company. If the suppliers represented in the market do not further develop their technologies, new market participants can generally threaten the market position of the established suppliers.[40]

  In this context, it is also possible to enter small market segments or to enter the market by offering additional components for existing software packages. Because established suppliers, whose offerings are designed to cover the widest-possible user base, are rarely able to respond to special needs, market shares remain that can be occupied and expanded by alternative suppliers.

- **Great significance of standards and consistency**

  The combination of low marginal costs of production and network effects on the demand side can lead to the emergence of de facto standards.[41] For suppliers of complementary products as well as for users, established standards create a reliable technology platform. Software suppliers can be assured that software that has been programmed according to certain criteria is compatible with a defined standard. Users benefit from a software standard by knowing that applications and hardware components are compatible with a given standard.

- **Great significance of patents and licenses**

  The ability to easily copy software makes it difficult to exclude users. Only the granting of patents and the licensing of software creates the legal framework for establishing and enforcing ownership rights. They are requisite to the excludability of software. As the discussion on the public-good attribute of software has shown, software would not be developed within the scope of market processes without fixed ownership rights. The ability to easily copy a software product would lead to a short-term price competition and to a loss of

[40] Cf. HOCH ET AL. (1999) and SCHMIDT AND SCHNITZER (2003), p. 7.

[41] Cf. OECD (2002), p. 105, and SHAPIRO AND VARIAN (1998), pp. 135 et seq.

market shares held by the initially innovative company. Fixed ownership rights in software is required by companies before they invest in any development of software. Investments can be financed through the sale of software only if the ownership rights in the software can be enforced effectively.[42]

[42] By purchasing the software, the buyer is not granted the full rights of disposal but the right of use is restricted to the areas stated in the license. The same applies to CDs or DVDs, the purchase of which authorizes private use only. Cf. EVANS AND REDDY (2002), p. 6 and p. 17.

2. Economics of Open-Source Software

2.1 Open-Source Basics

2.1.1 Attributes

The term "open-source software" was not coined until 1998.[43] The original idea of so-called "free" software, however, originated in the 80's. Even if both terms are used for the same form of software, the label "free" software emphasizes rather the liberal principles in dealing with software – "Free software is a matter of freedom: people should be free to use software in all the ways that are socially useful."[44] Further, restrictions in the use of the software were wrong and ultimately "all published software should be free software"[45].

The open-source initiative, on the other hand, takes a more pragmatic stand: the name open-source software had been chosen primarily in order to open up "free" software to more widespread interests, even commercial exploitation. Even if "open-source software" is a controversial term and sets other priorities than "free" software, it has been accepted in general usage and will be exclusively used below.[46]

Unlike proprietary software, the defining attributes of open-source software are the user's extensive rights of disposal of the software subject to the licensing terms and a voluntary self-organization of team development that is not based on work contracts.[47] The software user has the following rights in connection with open-source software:[48]

- The right to use the program for any purpose.
- The right to understand how the program works and how to adapt it to one's own needs.

[43] Schiff (2002) provides a brief outline of the literature available on the economics of open source.

[44] Cf. www.gnu.org/philosophy.

[45] STALLMANN (2001).

[46] One of the things criticized the most about the term "open-source software" is the fact that it aims solely at the technical design of software and neglects the original idea of free availability. Cf. GRASSMUCK (2002), pp. 231-232. The history and definition of free software and open-source software are presented in 2.1.2.

[47] See 2.1.3 for more information on how development is organized.

[48] Cf. www.gnu.org.

- The right to distribute copies to other users.
- The right to improve the program and to make the improvements available to the public.

From a programming point of view, the openness of the source code forms the basis for the development of open-source software. A user is able to contribute to the further development of software only if the source is open and the user is permitted to change the source code. Someone who knows the source code can develop new versions of the program or correct errors and improve the program.[49] This is in contrast to proprietary software, the source code of which is not principally – or only as machine-readable binary code – made available by the producer to the user.[50]

Table 2: Software Categories

			Technical property Disclosure of the	
			source code	**binary code**
Economic attribute	**Distribution is**	**free of charge**	Open-source software *Examples:* Linux, Apache	Freeware, public domain *Examples:* Adobe Acrobat Reader Pegasus mail
		subject to charges	Commercial open-source software *Examples:* Open-source software distributions	Shareware, commercial software *Examples:* Windows, MacOS

Source: BERLECON RESEARCH (2002c), p. 11.

Categorizing software by disclosure of the source text (source code, binary code) and the kind of distribution (free of charge/against payment) produces the matrix

[49] Cf. MORNER (2003), p. 318.
[50] Cf. BERLECON RESEARCH (2002c), p. 11.

shown in Table 2 that reflects the ideal types of software categories. According to it, the primary software forms are:[51]

- **Open-source software**
  The source code of software is available and the licensing terms allow the source code to be modified. The openness of the source code prevents commercialization. Programs using the published source code must again be subject to the licensing terms of the original source code. Known examples of open-source software are the operating system kernel Linux or the Apache web server.

- **Commercial open-source software**
  Software engineering under open-source conditions does not preclude the software from being used commercially. Open-source software, too, can be distributed in exchange for payment. However, this is impeded by the fact that, in principle, everyone is permitted to distribute the software free of charge. This group also includes commercial open-source business models that are based on open-source software and generate profit with complementary hardware, services or software. [52]

- **Freeware, public domain**
  In the case of freeware, the source is not available; it is delivered in binary form and may not be modified. However, freeware may be copied and distributed free of charge. [53]
  In public-domain software, the author waives any rights and claims. Consequently, the user has unlimited rights of exploitation. Open-source software, on the other hand, grants the user rights of exploitation, but expressly makes them subject to certain conditions. As public-domain software is often not distributed in source code, however, modification possibilities are limited.[54]

- **Commercial/proprietary software**

[51] Cf. BERLECON RESEARCH (2002c), p. 11. For more information on software categorization, cf. LESSIG (2002), pp. 52 et seq.

[52] This is dealt with in more detail in section 2.2.2 starting on page 41.

[53] Scientific publications can be compared to freeware. For the most part, they are distributed free of charge. Scientists consider reputation to be the compensation for their publication. However, the higher the commercial exploitability of the research findings (like in the field of natural sciences), the more restrictions their scientific publication is subject to. Cf. EVANS AND REDDY (2002), p. 7.

[54] Cf. SPINDLER (2003), p. 18, and HANG AND HOHENSOHN (2003), p. 7.

Commercial or proprietary software is normally only distributed without disclosure of its source code. Commercial software is distributed in licensed form. The license sets forth certain terms of use for the user that, as a rule, preclude multiple use or a circulation of the software. Because the source code is not accessible, it is technically impossible to modify the software.

These four software categories reflect ideal types of groups. In reality, these groups cannot always be separated that selectively. There are various financing methods and possibilities to view the source code as well. There is, for example, software that may be used free of charge but that calls for a voluntary payment in form of a donation. Shareware can also be used for a certain period of time, after which, however, it must be paid for. Even in the case of proprietary software that is distributed only in binary code, the source code can be disclosed. For example, within the scope of its Shared Source Initiative, Microsoft discloses some parts of its source code to customers, partners and governments.[55]

2.1.2 Beginnings and Licensing Models

In the early days of computing, there was no independent software market. Computers were mainframes deployed only in large companies or at special data centers at universities. These computers could only be operated by experts that were also able to make changes to the software or who developed their own software and shared it with other experts. Initially, software was developed exclusively for one’s own use, specifically for different types of hardware and as such highly individually. A mass market for standardized software could not exist, as the corresponding hardware was not designed for the mass market in terms of size and price.

Developers’ activities revolved around programming software for the computers they themselves used. They had strong incentives to share their work with other developers using the same hardware. Because software was distributed only as a necessary complement to hardware, and developers had to rely upon their mutual exchange, the source code of software was open and software was freely available. Even if the software was installed on different computers, it still featured clear

[55] Cf. NO AUTHOR (2003).
See www.microsoft.com/resources/sharedsource/Initiative/Initiative.mspx for an overview.

attributes of custom software. Though the programs were not subject to a charge, the programmers were paid for programming.[56]

In this development environment, the operating system Unix was programmed in 1969 by two employees of the telecommunications group AT&T. The Unix source code was open and could be further developed by other developers. Without being promoted or advertised by a company, Unix spread quickly. After the telecommunications monopoly of AT&T split up, AT&T began marketing Unix independently. By that time, different, incompatible Unix versions had been created because the source code was freely accessible and modifiable.

One of the first licenses for free software was the **Berkeley Software Distribution License** (BSD license). At UC Berkeley, programs were developed on the basis of UNIX and distributed as Berkeley Software Distributions. The BSD license was established so that the program elements produced at UC Berkeley could be distributed independently of the AT&T license. It permits the use, modification and free distribution of the source code or binary code. The only condition for further use is that any changed and distributed programs must contain a copyright notice referring to the University of California.[57]

Though the BSD license does stipulate that any software based on it comes under the free licensing terms, i.e., that it may also be modified, it does not, however, expressly state that the modified software must also be available in source code. As such, the actual modification possibilities are limited, and it is easier to establish a cost-based distribution of programs based on the BSD license.[58]

In 1984, the commercialization and assertion of ownership rights in software which were opposed to the originally free programming, modification and distribution of software, led to the GNU project. This was the actual beginning of software development whose engineering and distribution was deliberately classified as free. The objective was to write an operating system similar to Unix, with an open source and capable of being further developed in voluntary cooperation. To distribute GNU software, the Free Software Foundation (FSF) was established in 1985. In order to protect future free software from commer-

[56] Cf. GRASSMUCK (2002), pp. 202 et seq. for more historical information.

[57] Cf. HANG AND HOHENSOHN (2003), pp. 26-27, EVANS AND REDDY (2002), pp. 8-9. The copyright notice requirement was waived in 1999.

[58] Cf. GRASSMUCK (2002) pp. 216-217 and pp. 279-280 and SCHMIDT AND SCHNITZER (2003), p. 5.

cialization, the Free Software Foundation published the **General Public License** (GPL) in 1989. This license extends the four rights shown in section 2.1.1.

GPL's copyright protection is necessary to prevent the software from being used commercially. Without the license condition stating that any software using the source code must also be subject to GPL terms, proprietary software suppliers would be able to integrate the free source code in their software free of charge. This requirement of GPL protects the intellectual property of the free software and, in return for its use, requires that the source code be freely accessible as well.[59] Consequently, the use of software is not principally free, but only to the extent as it stays within the limits stipulated by the licensing terms.

An important component of the GPL is the so-called "virus effect". It arises out of the following requirement: "*You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the program or any part there of, to be licensed as a whole at no charge to all third parties under the terms of this license*."[60] The Free Software Foundation uses, contrary to "copyright", the term "copyleft" as a name for this method of making software programs freely accessible and preventing their commercial distribution. As a result of the copyleft requirement, software that is partially based on software released under GPL conditions must also come under GPL conditions. This precludes any commercial use of software or of source-code elements in other programs, that is, the generation of profits from selling the programs.[61]

A variation of the GPL is the LGPL (**Library General Public License**, as of 1999 Lesser General Public License).[62] Under this license, programs linked with a library that is subject to LGPL do not have to be considered derived products as defined by the GPL. The use of libraries within a program obscures the lines between use and modification, and it could not clearly be determined in which case accessing a GPL library required that the derived program be subject to GPL. LGPL permits this use of libraries without causing the derived program to have to be subject to GPL.

[59] Cf. LERNER AND TIROLE (2000), pp. 5-6 and SCHMIDT AND SCHNITZER (2003), p. 5.

[60] Cf. www.fsf.org/copyleft/copyleft.html.

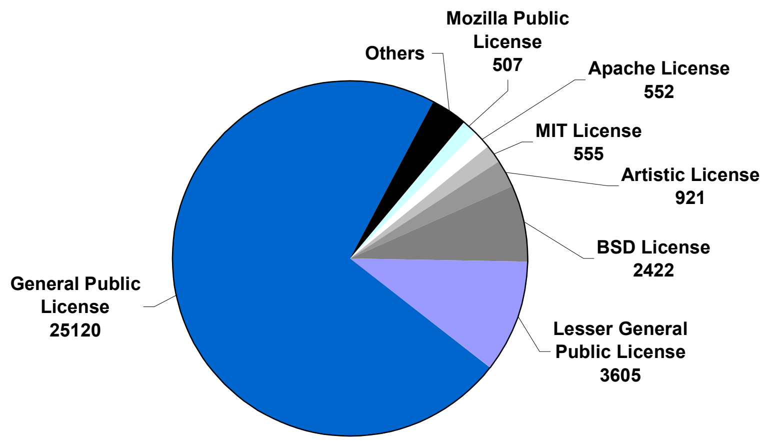
[61] However, please note the possibility of "dual licensing". In dual licensing, the holder of a copyright can publish software under the GPL terms in general and provide this software to certain users under conditions other than the terms of the GPL. However, FSF handles this possibility very restrictively. See www.fsf.org/copyleft/gpl-faq.html.

[62] Cf. GRASSMUCK (2002), pp. 289-293 and HANG AND HOHENSOHN (2003), p. 26.

Software released under LGPL thus offers better options in certain cases for combining free and commercial software.

As open-source programs continued to spread and be developed, the licensing method changed. While the GPL, a very restrictively interpreted software license, dominated the 80's, the 90's saw a trend towards more flexible, less restrictive licensing agreements. The use of more liberal licenses was supported particularly by developers advocating commercial activities and the use of proprietary code in certain segments in order to guarantee a broader provision of open-source software.[63]

Illustration 4: Distribution of Licenses



Source: BERLIOS (2003).

In the late 90's, an attempt was made, as mentioned above, to establish the term of open-source software as an alternative to free software. In 1997, the **Open-Source Initiative** (OSI) was established. The open-source definition developed by OSI is not an independent license but a quality seal for rating licenses. If a license satisfies the criteria stated in the open-source definition, it may bear the protected title of "open source".[64]

[63] Cf. LERNER AND TIROLE (2000), p. 7. See also LERNER AND TIROLE (2000), p. 30 for more on the problems that may arise from more liberal licenses.

[64] Cf. MENDYS-KAMPHORST (2002), p. 10 and HANG AND HOHENSOHN (2003), p. 14.

The strategy pursued by OSI entailed popularizing open source and countering the anti-business image of free software and of the Free Software Foundation.[65] These considerations led to the establishment of the term "open-source software" as an alternative to free software. OSI publishes a list of major licenses on its website and states in a catalog of criteria when a license can be rated as open source.[66]

In addition to the licenses described here, there are many different licenses that were designed for certain projects.[67] Illustration 4 shows the distribution of the licenses in the Sourceforge.org database as of January 2003.[68] 89 percent of the projects listed there are licensed under GPL, LGPL or BSD.

The most important or most popular licenses are listed in the following table. Freeware is stated only for comparison purposes with open-source software.

Table 3: Major Licenses and their Essential Terms

	Use free of charge	**Source code modifiable**	**Source code must be open in derived products**	**Combination with proprietary elements not possible**
Freeware	X			
BSD	X	X		
LGPL	X	X	X	
GPL	X	X	X	X

Source: With reference to SPINDLER (2003), p. 19 and BERLECON RESEARCH (2002c), p. 16, modified.

[65] A popular comparison for illustrating the image of free software is that of "free speech" to "free beer". Many companies were inclined to associate the term "free software" with "free beer" rather than with "free speech" and were therefore reluctant to introduce an operating system that was generously given to everyone free of charge.

[66] www.opensource.org/licenses/ and www.opensource.org/docs/definition.php. Cf. also WEBER (2000), pp. 10 et seq.

[67] For an overview and categorization of the different licenses, see www.ifross.de/ifross_html/lizenzcenter.html.

[68] Sourceforge.org provides provides a database with open-source projects. An examination of the various licensing models at sourceforge.org can be found in LERNER AND TIROLE (2002).

2.1.3 Work Organization and Development

Team development in engineering open-source software may take place within various organizational structures. As a rule, software developers voluntarily join an open-source project and, depending on their contribution, are involved more or less deeply in the software project and its decision-making structures. These software projects come about spontaneously and organize themselves. Participation in a software project is voluntary and does not entail any financial rewards.[69]

Unlike traditional software development, open-source projects do not start out by identifying the customer's needs, but are mostly sparked by an idea or a specific problem that developer has.[70] An open-source project starts with the publication of the project and the source code. If the project seems interesting, more developers show up and join the project. If a project grows, structures may develop in which a central maintainer or a core team makes essential decisions concerning the future development. From larger projects, modules may spin off for which there are again responsible maintainers.[71]

Under a maintainer or core team, there are many ways to participate in a software project, from simply testing the program and offering tips for new functions to bug-fixing and the actual work on the source code. Communication takes place via e-mail, mailing lists and news groups.[72]

The comparison between a bazaar and the construction of a cathedral is often quoted as an example to illustrate this kind of team-based development. In this comparison, the development of open-source software is likened to a bazaar, while proprietary software is seen as the construction of a cathedral. Open-source software is said to be a "great babbling bazaar of differing agendas and approaches out of which a coherent and stable system could seemingly emerge only by a succession of miracles".[73] Unlike this "bazaar", the development of proprietary software is understood as a "cathedral" and as such as the epitome of hierarchically structured division of labor marked by strict authority.

[69] Nota bene: nonremunerated participation pertains to the engineering of generally usable packaged software. Programming custom software is also possible on the basis of open source. In that case, payment is made for the developers, not for the software. This will be dealt with in more detail at a later point.

[70] Cf. SMITH (2002), p. 72.

[71] Cf. MENDYS-KAMPHORST (2002), p. 13.

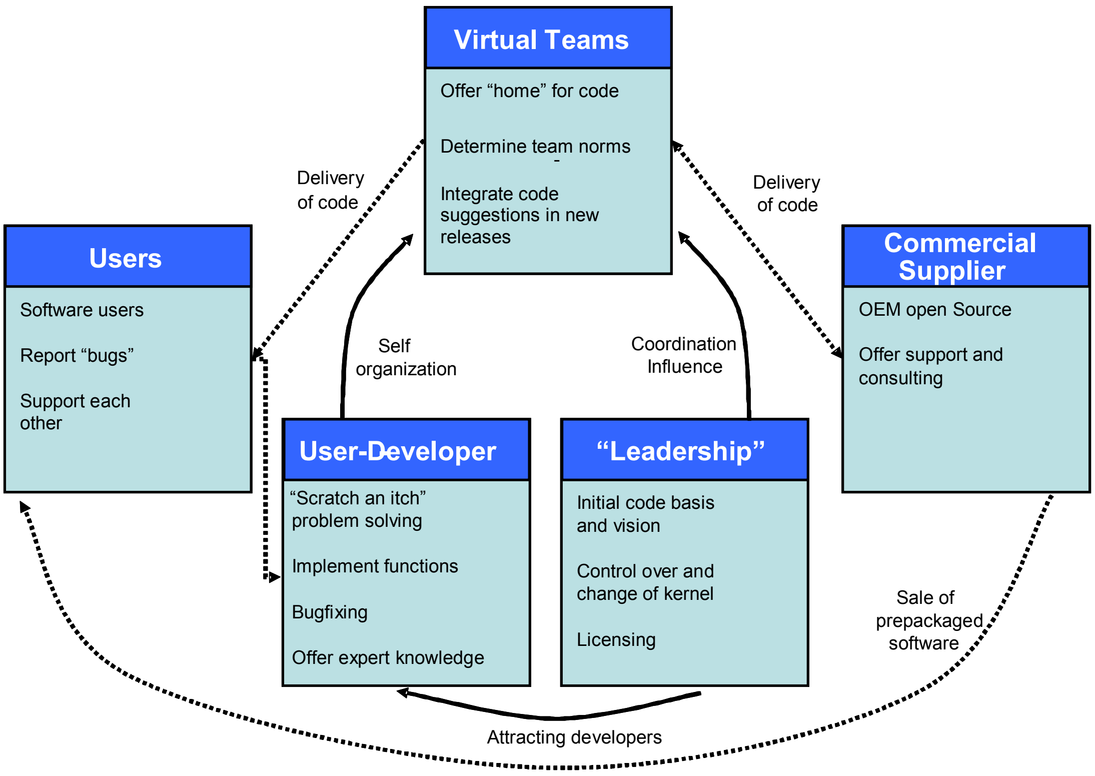
[72] Cf. MORNER (2003), p. 320.

[73] Cf. RAYMOND (1998a).

However, this form of voluntary development association also carries with it the risk that projects may "dry up". If developers turn to new projects that, in their view, are more interesting and possibly higher-profile, projects may not be carried through to the end due to a lack of support. As an example, of the programs currently registered with sourceforge.net, only 16% reach the development status "production/stable", and only 2% reach the status "mature".[74]

Open-source software must not necessarily be developed in this self-organized fashion, however. Open-source software can also be written in companies by in-house developers. For example, distributors often employ in-house developers to make changes to specific development levels of software. In this case, it is essential that all other users of this software also have access to the contribution made by a company's in-house developers.

Illustration 5: Schematic of How an Open-Source Project Works



Source: BOSTON CONSULTING GROUP (2002), p. 8, authors' modifications.

[74] www.sourceforge.net lists development status as follows: 1. Planning, 2. Pre-Alpha, 3. Alpha, 4. Beta, 5. Production/Stable, 6. Mature and 7. Inactive. On October 27, 2003, the following information was found: 1. Planning 13,248 (26%), 2. Pre-Alpha 9,388 (18%), 3. Alpha 8,833 (17%), 4. Beta 10,739 (21%), 5. Production/Stable 8,503 (16%), 6. Mature 823 (2%), 7. Inactive 327 (1%).

This kind of networked, collaborative work done on different subproblems is only possible with a modular project structure. As a result, the functionality of programs often pertains to individual subtasks, and only the interplay of different of modules and programs produces the desired functionality.

Illustration 5 outlines the teamwork process within an open-source project as a flow chart. It must be noted at this point, however, that software users only have an opportunity to introduce their ideas to the software if they themselves become developers. There is no feedback channel within the open-source model that nondeveloper users could use to voice their needs. On the proprietary software market, they voice their needs with their purchase decision.

However, open-source projects can also grow out of programs that were initially of a proprietary nature. In 1998, for example, Netscape opened up the source code of its internet browser "Netscape Communicator", hoping to accelerate its browser's development by involving independent programmers and to re-gain lost market shares. The software package Star Office, which was initially sold as proprietary software and whose source code was later released, followed a similar path.

In the case of Netscape, not the entire source code was released, but only individual parts. In addition, Netscape first insisted on licensing terms that later would have made it possible to re-appropriate the released code. This reduced the willingness to cooperate in the Mozilla project.[75] As a result, it was mostly paid employees who worked on the Mozilla project; no more than two dozen external developers are said to have been involved in the project. In this particular case, the willingness to participate may also have been curbed because the project was headed up by a for-profit company.[76] The release of a stable version was only possible after adjustments were made to the new licensing structure; all told, this took four years.[77]

Another problem with open-source projects that develop from former proprietary software is that a developer community has to be found for a defined and finished source code. In this situation, it is much more difficult to become familiar with the source code of a finished product and then to work with that code than to independently develop one's own project in which the entire history of its

[75] Concerning this, see: www.gnu.org/philosophy/netscape-npl.html.

[76] Cf. LERNER AND TIROLE (2000), p. 28.

[77] Cf. OSTERLOH, KUSTER, AND ROTA (2002), p. 15.

development is made transparent.[78] Another important consideration concerning these open-source projects is that the source code of proprietary software is only opened up for the "use of leftovers" wherever there were not enough users who were interested in the proprietary version and, as a result, the software was a commercial failure.

2.2 Business Models and Market Overview

2.2.1 The Value Chain of Software - A Comparison of Open-Source Software and Proprietary Software

The differences in the development process of proprietary and open-source software lead to differences in how the two models cover the software value chain. In the value chain, software development is followed by software services which include consulting, implementation, support, training and application management.[79]

And though the individual elements of the software value chain are identical for packaged and custom software, the sequence in which they are applied differs. For example, in contrast to packaged software, consulting precedes the actual programming process in custom software. There are also differences between the value chains of proprietary and open-source software. Even though all the value-chain elements can be found in both, there is a different weighting of the individual steps in proprietary and commercial open-source business models. Furthermore, it is not possible to generate profit with individual elements of the value chain in the open-source model. This will be dealt with in more detail later on in this study.

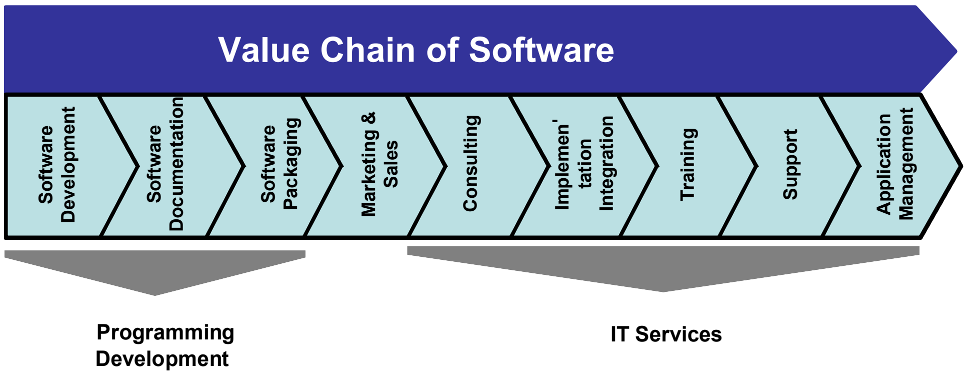
The development of software is the first element in the value chain. In proprietary software, determining the software specifications based on the projected customer wants is part of software development. This step further entails specifying the design of the software and writing the source code. The production is followed by the documentation and packaging of the software. In the packaging process, individual software products are grouped to form a marketable package. Software development, documentation and packaging together constitute the production of software. While the documentation and packaging are only one component in the

[78] Cf. GRASSMUCK (2002), p. 257.

[79] More on this and on the value chain as a whole, cf. BERLECON RESEARCH (2002c), pp. 22-29.

production process of proprietary software, individual open-source business models (distributors) focus solely on these elements.

Illustration 6: The Value Chain of Software



Source: BERLECON RESEARCH (2002c), p. 23.

It is the job of marketing to optimize the use of its tools within the marketing mix and so to promote the visibility, the acceptance and thus the sales of the product. Differences between proprietary software and open-source software are evident in all elements of the marketing mix. Overall, the open-source model does not offer as many opportunities to influence the product and sales through marketing as does proprietary software.

There is no scope for influencing the product design – the first element of the marketing mix – because of the unique open-source development process. In the case of proprietary software, the developing company can determine the time of market entry, quality or certain customer requirements, just to name a few aspects. There are also significant differences between open-source software and proprietary software with respect to price, which is another marketing tool. Open-source software can be priced only if, for example, another functionality, in addition to the program, can be added through packaging. Because the software could be bundled independently or because a software bundle that has been sold can be used on different computers, there is much less pricing leverage available in open-source software distributions than in proprietary software.

Distribution and advertising are further elements of the marketing mix. The bundling and distribution of a certain software package is the key element in marketing open-source software.

Marketing is followed by consulting, which is a component of software services. In consulting, a company's unique situation is analyzed, its software requirements are determined and the software is chosen. There are no differences between open-source software and proprietary software in this regard.

The software is installed and tailored to the given requirements during the implementation process. Among other things, compatibility within a network or different applications must be taken into consideration. Compared to proprietary standard packages, open-source software offers nearly limitless adaptability in this respect.

Users are familiarized with the installed software in training courses. There are no fundamental differences between proprietary and open-source software concerning the training provided.

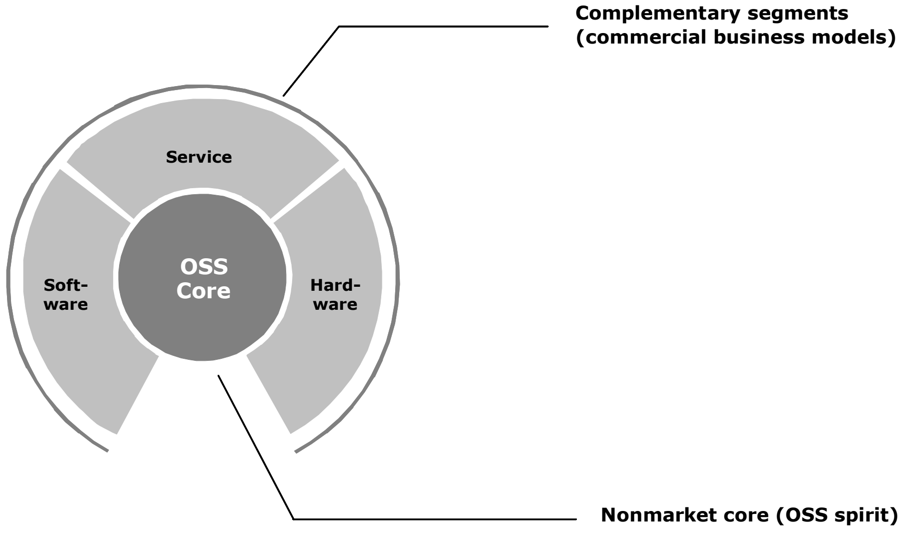
Support is necessary if the user has problems using the installed software. Suppliers of proprietary software offer a wide range of different types of support, from FAQs and newsgroups to in-house hotlines. There are also many possible sales models with various forms of support. Support for open-source software is offered primarily by the community that engineered the software. At the same time, distributors and independent consulting firms also offer open-source software support.
The final element in the value chain is application management. It comprises all the elements that are required to properly operate the software and also includes possible updates or backups. Application management can be handled by system administrators within a company or may even be outsourced.

2.2.2 Open-Source Software as a Basis for Business Models

An important factor in the development of free software is to prevent anyone from acquiring the ownership rights in the software and as such from using it commercially. The related licensing models have this goal as well. The fact that open-source software may be legally copied and, for example, downloaded from the internet precludes the commercial use of software through cost-based licensing models. Consequently, there is not a market where suppliers and users of open-source software can meet.

Commercial business models based on open-source software have emerged nevertheless. They all build on open-source software and use that as a basis for their own additional services or products. All business models attempt to increase the demand for their own complementary product through their involvement in open source. From a business point of view, investments in open-source projects are only profitable for companies if they can generate profit with other services or products because of these investments. Because improvements in open-source software cannot be used to directly generate profit, companies must profit indirectly, through other products and services, from open-source development. Open-source business models can therefore be described as indirect business models because sales are not generated with the actual product but with additional products and services based on open source.[80]

Illustration 7: The Open-Source Core and Indirect Open-Source Business Models



.

Although the profits can be generated only indirectly, it must be emphasized that, in contrast to the developer community, companies have purely financial motives for participating in open-source projects. Illustration 7 shows different ways of designing commercial business models based on open source. A distinction is made between business models offering additional services, additional software or additional hardware. All these models draw on a pool of software and developers and add their own products and services to open-source software. There are, however, only limited incentives for companies to directly invest in the development of software. The investments in open-source software cannot be financed with the profits from the sale of this software. For that reason, all expenses incurred in development must be generated through additional offerings.

[80] Cf. RAYMOND (1998b), LERNER AND TIROLE (2000), p. 26 and SCHMIDT AND SCHNITZER (2003), p. 12.

The individual business models and the limiting factors are discussed in more detail below.[81]

2.2.2.1 Selling Additional Services

Distributors of open-source software offer the bundling, testing and adaptation of that software as an additional service. Known Linux distributors are, for example, Red Hat, SuSE or MandrakeSoft. The end user can purchase different Linux software bundles for various purposes (e.g., server applications, desktop applications, software for administrators or developers). The various versions of different distributors are not necessarily intercompatible.

Distributors draw on the existing open-source software pool and, by bundling and adapting the software, perform a task that, for proprietary software, is handled by the developing company. A Linux distribution contains the Linux kernel and numerous additional components that jointly form the Linux operating system. For in-house versions to be developed, first the latest versions of individual components are collected (which is called packaging) and then tested, adapted and optimized. Finally, the distribution is documented and set up in a way that facilitates easy installation. The fact that distributors are necessary for making adjustments and adaptations to the software prior to it being usable by a wide circle of users is also a sign of the inability of the open-source development process to bring forth products that end users can use immediately.[82]

A buyer of a Linux distribution no longer needs to search for the software, download and then adapt it so that the individual components operate together flawlessly. Along with a distribution, a buyer also purchases the opportunity to receive updates or bugfixes in certain sequences. In principle, any user could also bundle and install these components and fix any bugs independently.

Distributors employ their own developers who adapt the software as required. Again, every development contribution made by a distributor to his or her own Linux distribution must, however, be made freely accessible. Though Linux distributors save the bulk of the development efforts in the distributed software and benefit from a further distribution of the software, their contribution to the development not only benefits them but all the other companies that are active in

[81] BERLECON RESEARCH (2002b) offers a comprehensive overview of the open-source software activities of different companies.

[82] Cf. LERNER AND TIROLE (2000), p. 26.

this market as well. As a result, their development contribution also directly supports potential competitors who, as freeloaders, also benefit from the development investments.[83]

Distributors have very little leeway in pricing their products. Not only can users themselves bundle the individual software components for free, buyers can also pass on the distributions that are sold free of charge. Likewise, other companies can take over a given distribution and distribute it as their own distribution. If the source code is disclosed and visible to any recipient, then the software can very easily be distributed further. This exerts pressure on prices until the selling price reaches the level of the average sales costs.[84] In this case, the software distributor's average sales costs are the lowest price level because, provided the distributor does not incur any development costs, the distributor merely has to bear the sales costs. Accordingly, it is very easy for users to have free access to the distributed product and for other suppliers to enter the market, which severely restricts the distributors' pricing leverage.[85]

Because of the limited opportunities to generate profit with the sale of distributions, many distributors also offer consulting, implementation and training services. In this aspect, their engineering know-how acquired by bundling the software is very useful to them. They are, however, competing with established IT consulting firms that handle the bundling of software and its installation according to the specific requirements of their customers.[86] In this regard, IT consulting firms are not restricted to open-source software offerings but can choose the best-possible, platform-independent solution for their customers from proprietary or open-source software.

2.2.2.2 Selling Additional Software

Business models based on the sale of additional software use open source as a starter and basis for selling complementary proprietary software products or even for offering support services. Companies that have implemented this business model often have a very close relationship with individual open-source projects. In many cases, the company founders are also the initiators of the projects, or they

[83] Cf. LERNER AND TIROLE (2000), p. 27.

[84] Cf. SCHMIDT AND SCHNITZER (2003), p. 3.

[85] Cf. EVANS AND REDDY (2002), p. 37.

[86] Cf. BERLECON RESEARCH (2002c), p. 43.

have assumed important developer functions in which case it is possible that the principal developers of that particular software are employed by the companies. Nevertheless, even these projects are dependent on the support of the developer community. The more commercial the focus of a company, the more difficult it becomes, however, to attract enough volunteer developers, as the commercial focus is contrary to the developers' interests, which are geared towards voluntary contributions and free availability. An illustration of this fact is the aforementioned example of Netscape.[87]

The basis of these business models is freely available open-source software on which additional add-ons or programs with enhanced functionality are built and which are then distributed for a charge. Furthermore, the dual-licensing strategy allows making a product freely accessible as open-source software and selling that very same product as a cost-based version to companies wanting to use it in combination with other, proprietary software.

For example, MySQL offers its database free of charge as open-source software. There is also the option, however, to purchase the software under a specific corporate license so that it can be deployed together with other, proprietary software.[88] Ximian offers software that in terms of looks and functionality closely resembles Microsoft Outlook. A cost-based component can be purchased as an add-on to this software, which allows it to operate together with a given e-mail server.

However, it is possible even for commercial software providers like Oracle, SAP or IBM to sell additional software. These providers have generally reached a significant market position in the proprietary software market with certain software products. Adapting their software to other platforms enables them to broaden their potential installation base. In this regard, it is important that the open-source platform can be used as a free base. If a software user is not required to additionally invest in a proprietary platform, the potential profits on the application level are correspondingly higher. It should be noted that the software offered by proprietary software providers for an open-source platform is not necessarily open-source

[87] Cf. 2 section 2.1.3.

[88] See www.mysql.com.

software in itself.[89] This software is merely open-source compatible and uses open source as an operating system platform.[90]

2.2.2.3 Selling Additional Hardware

Companies can also support open-source projects to promote the sale of their own hardware products. IBM is the best-known example for this kind of strategy. IBM, however, is not only a hardware firm but also one of the world's biggest software producers. By IBM's own account, the company had invested 1 billion U.S. dollars in various open-source projects by 2001, which included adapting Linux and Apache to the different IBM hardware platforms.[91]

IBM uses adapted Linux versions to create a uniform operating-system basis for different server platforms on which the different IBM software components can then be executed. The software is therefore tightly integrated into the hardware and makes it possible to create uniformity within the full range of hardware products. If the operating system platform is considered a fixed hardware component, profits can then be generated with the sale of the hardware. This is particularly attractive for hardware firms because of the fact that the user does not incur any extra costs for purchasing an operating system. If the user has fewer software expenses, however, the hardware firm's pricing leverage increases.

Open-source projects are also supported by Intel. Intel's primary interest is to create software that is highly compatible with its hardware products. For that reason, the support can be viewed as a promotion of the company's own products rather than as an independent business segment.

2.2.2.4 Assessment of the Business Models and Limiting Factors

All commercial open-source business models have to function in the open market. Companies invest in the development of open-source software only if they can offer additional services or products with which their investments can be financed. In these business models, open source, as it is available free of charge, becomes an attractive input factor with which to attempt to generate additional profits with services building on open source. Freely accessible software remains the basis of

[89] The GPL applies only if new software uses components of other software licensed under GPL ("viral effect").

[90] For an economic model on the support of open-source software by proprietary providers, see Mustonen (2002).

[91] Cf. Evans and Reddy (2002), p. 34.

these business models, however, limiting the opportunities to build stable revenue models as well as the incentives for investing in open-source projects.

Concerning consulting, training and support offerings, open-source business models are no different from business models in the proprietary world. The same applies to the distribution of specific literature or the offering of trade fairs and magazines.

It is crucial to the success of commercial open-source business models that the additional offering cannot be easily offered also by another company – which requires that ownership rights in this additional offering can be enforced. The distributor example has shown that the market barriers for other companies are very low in this area. Based on the work performed by the distributors, similar software packages or support offerings can be offered by other companies as well. If, however, the actual source of profits for open-source projects consists of additional products in which companies can enforce ownership rights, that serves more as proof of the superiority of proprietary business models than of the stability of commercial open-source business models. Commercial open-source business models seem to be especially successful when they are as far removed as possible from the open-source world, and when the ownership rights in the additional services can be easily enforced. This means that a provider of a complementary service has the ability to exclusively market his or her offering. There is then no pressure on prices as other providers cannot easily copy the offering. In that, however, successful commercial open-source business models are no different from proprietary business models, either.

As to specific licenses for proprietary open-source software versions or components, the additional components can be marketed separately/exclusively. Hardware firms can also consider open-source software a simple way of equipping their hardware with a higher level of functionality. Because of the great exclusive marketing opportunities of hardware products, the integration of open source seems comparatively easy.

Various factors limit business possibilities within a commercial open-source business model. The more open-source business models commercially market additional products, the more difficult it might become to find sufficient voluntary (and from the company’s view, free) support in the developer community for a project. This, however, increases the need to invest in building up a permanently employed development team. Yet the work of these developers would then also be freely available to all other users as well. The higher a company's investments in development, the likelier are incentives to protect these investments and to restrict free accessibility, for example through specific licensing models. Therefore, with increasing commercialization, these models move ever further away from the world of open source and align themselves with proprietary models.

Another limiting factor is a company's limited ability to set itself apart within the open-source development environment. The fact that the results of programming activities are also available to all other potential and actual providers makes it very easy to copy not only the product but also the business strategy. Consequently, the first supplier of a certain product does not have greater opportunities for generating profits for too long. If a business model is based on an open-source product, the market position of that product can easily be contested by imitators (fast erosion of pioneering advantages).

2.3 Consequences of the Lack of Market Coordination in the Open-Source Model

2.3.1 Has Software Ever Been Free? The Development of the Software Market from an Economic Perspective

Accounts of the history of open source often point out that in the beginning, all software was free. This refers to the aforementioned kind of software development prevalent when computers were beginning to spread. At that time, specialists and system administrators developed software for mainframes and provided other users with the changes and further developments they had made. The developers were employed by universities or large companies from whom they received their pay. As the market for specific software developments was very small, the employers of these developers rarely showed an interest in marketing the software externally. Mainframes and the necessary software were primarily seen as a tool and input factor and not as a marketable product.

A determining factor in the willingness to open up one's software developments to other users and to distribute the software free of charge was the fact that the developer group and the user group were by and large identical. As mainframes were not widespread, every developer had an incentive to open up his or her development contribution to other users as it was safe to assume that these other users would also enhance the software. This allowed a bartering system to emerge based on development contributions. As long as there were no or only a few users that did not contribute to the development, there was no danger of freeloaders who would benefit from the software development without contributing to its enhancement. As such, however, software was not nonmarket even at its inception; it was merely distributed by barter. In this barter, one developer's programming contribution was exchanged for the programming contribution of another developer.

In the framework of the public goods theory, the development of software at that time can be classified as a club good. There was no rivalry in consumption, and excludability was indirectly guaranteed by the fact that only those people used software who could also enhance it. This led indirectly to the development of a club, one whose membership was restricted to users who were also developers.

The software developer/user identity disintegrated with the further spread of computers. Initially, software was designed as custom software or reproduced in small numbers for specific business applications. It was at that point that software users started to pay for the software instead of contributing to its development.

With the emergence of a mass market for personal computers, there eventually was a large number of users that did not make their own development contributions. The implicit developer-user club disintegrated. Without established ownership rights, there was not much willingness to make a development contribution that could then be used by many others free of charge. In this interim phase, software can be viewed as a pure public good, for which there is neither rivalry nor excludability. Only the definition of ownership rights re-established excludability and thus provided incentives to invest in the development of software.

There was also the fear that establishing the Free Software Foundation would give users access to freely developed software without the users providing their own development contribution. In that case, however, the use pertained to other companies that might integrate free software into their proprietary products, but would not open them up. In connection with the development of free software, the Free Software Foundation itself refers to a "club" that – along with the use of the developed software – is only open to those who open up their software as well.[92] The GPL was designed to allow just that. This ensured that nobody would use the software code for his or her own software products without opening up his or her

[92] Cf. FREE SOFTWARE FOUNDATION "The GNU GPL and the American Way" www.gnu.org/philosophy/gpl-american-way.html

own development contribution to others. The GPL thus acts as an exclusion option for those companies that do not open up their software.

2.3.2 Economic Motives for Participating in Open-Source Projects

As to the motives of open-source developers, one commonly raised argument is that there is a "gift culture" within the open-source movement that is compelled by altruism and reciprocity. According to this argument, developers contribute to open-source projects because they enjoy being part of the community and consider their development contribution to be compensation for the programs and support from which they themselves have benefited.[93] Against this backdrop, the exchange of "goods for money" is to be superseded by an exchange of "gifts for reputation".[94] It is unclear, however, why the motives of altruism and reciprocity should be more important in the development of software that in other fields. The motive of altruism as a driving force seems to be an inappropriate explanation especially as it is not other private users who benefit from the programming work, but large companies. Relationships based on a mutual exchange are important mainly in small, manageable groups.[95] However, it is questionable to argue that reciprocity can also explain the programming contribution in a very large and anonymous group.[96]

In 2002, the Boston Consulting Group surveyed the developers registered with SourceForce.org to examine their motives. In this survey, the respondents provided the reasons listed in Illustration 8 as their motives for work.[97]

Motives like "intellectually stimulating", "work with team" or "nonwork functionality" can be viewed as motives that apply to other leisure activities, too. Ideological motives like "code should be open" are found in a similar form in other leisure activities. As a result, open-source development can be viewed as a normal form of leisure activity, the results of which are available to other users as well.[98]

[93] Cf. Raymond (1998b) and Schmidt and Schnitzer (2003), p. 10.

[94] Cf. Raymond (1999), Kollock (1999) and Franck and Jungwirth (2002), p. 124.

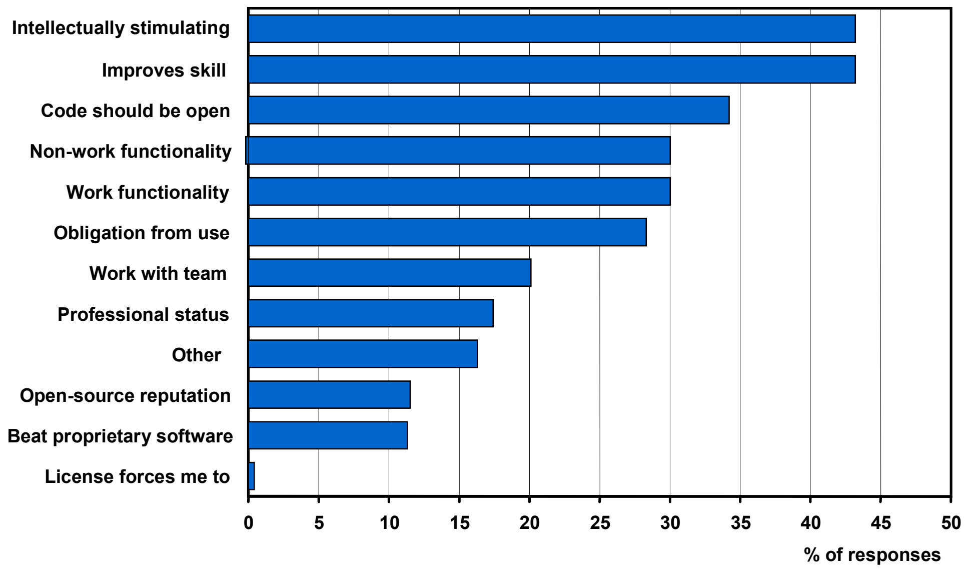
[95] Fehr and Schmidt (2002) provide an overview of economic research on the topic of fairness and reciprocity.

[96] Cf. Schmidt and Schnitzer (2002), p. 10. Lerner and Tirole (2000), p. 18.

[97] Cf. Boston Consulting Group (2002), p. 16.

[98] A similar effect is found in other clubs such as garden clubs, where the property is also open to the public. In this case, too, the leisure activity yields a positive utility for visitors.

Illustration 8: Motives of Open-Source Developers



Source: Boston Consulting Group (2002), p. 16.

However, there are also numerous additional motives that are directly or indirectly related with a professional activity. These include, for example, "improve skill", "work functionality", "professional status" and "open-source reputation". These economic motives for participating in an open-source project are examined in further detail below. In this examination, a distinction is made between the importance of open source in revealing one's own development skills, small contributions and the use of open source for continuing education.

According to economic criteria, a developer will only participate in a project if this activity is associated with a positive utility for that developer as compared with his or her costs. The developer incurs costs because of the time invested in the programming activity. The value the developer attaches to these costs depends on how much the developer enjoys his or her work. The developer can derive utility from a direct or postponed consideration.[99]

Payment is the direct consideration for the developer in a proprietary software project. A developer's direct consideration may also be that a software problem personally affecting him or her is solved directly. This may be the provision of a certain driver that is then jointly developed, the tailoring of software to the developer's specific needs, or a program bugfix.

[99] Regarding the cost-benefit considerations, cf. Lerner and Tirole (2000), p. 14 and Mendys-Kamphorst (2002), pp. 19-20.

If a developer makes only **small development contributions** within an open-source project, these contributions are associated with little effort and low costs for a highly qualified developer. This may include solving a simple problem, customizing software or developing a small add-on application. The ability to access the source code makes it relatively easy for these developers to make improvements that can also benefit others in their daily work. The opportunity costs of sharing the new code with others are low as well. If, in addition, the new developments are relatively unsophisticated, it is not worth the effort to protect this innovation. Moreover, the internet provides an efficient and inexpensive way to make this innovation available to the public. Consequently, the individual developer incurs only minimal costs in making and disclosing a development contribution.[100]

In addition, open-source development work can be used in **continuing education**. In this case, the developer uses his or her work in an open-source project to find solutions to questions in his or her professional life. The developer therefore receives a direct consideration for his development contribution in the form of the solution to the problem. Yet programming work as part of continuing education can also be viewed as an investment in the developer's future career. The consideration to be received in the future would in this case consist of better career prospects.

Another future consideration may consist of developers being able to **show their programming skills** (signaling) and to have these evaluated. Working in an open-source project thus sends out signals regarding the quality of one's own work, which can then be profitably used in a secondary market (labor market). As such, development work is used for signaling and to build up a reputation.[101] Similar behavior can be observed in scientific publications.[102]

Revealing one's own development skills and building up a reputation can be a major motive wherever development work goes beyond simple small contributions. As previously described, the bulk of development work in larger projects is performed by a small group of developers. By making a considerable development contribution, these developers can build up their reputation and use it in other areas. If building up a reputation is an important argument for a development

[100] Cf. SCHMIDT AND SCHNITZER (2003), pp. 10-11.

[101] Cf. FRANCK AND JUNGWIRTH (2002), p. 127.

contribution, then the bigger the proprietary market in which the reputation gained in open-source development can be used, the greater the incentive to participate in an open-source project. A big proprietary market means that there are more companies that would compensate developers for their commercial development work.[103] There are various factors influencing the scope of the signaling effect:

- The signaling effect is strong if a developer's skills can be evaluated by a large audience. This may lead to developers preferring such projects that attract as many other programmers as possible. Large projects in which many developers are already involved are therefore more attractive than small niche projects, whose further course and significance are yet uncertain. A resultant positive effect is that there is a tendency for better-quality work to be performed in high-profile fields. With regard to resource allocation, however, all developer capacity is concentrated in only a few fields, while other areas are not dealt with (on resource allocation, cf. 2.3.5.2).[104]

- The signaling effect is also stronger in those cases when the programming task to be performed is particularly challenging and if the developer group is able to assess and evaluate the development contribution. It is especially significant in "*settings with sophisticated users and an audience that can appreciate effort and artistry and thus distinguish between merely good and excellent solutions to problems*".[105] A key aspect in this context is the so-called "peer review". This means that one programmer's work can be reviewed by other professionally qualified programmers. A certain reputation can only be built with the help of "peer review", as it represents the only benchmark for the quality of a product. The incentives leading to signaling may therefore focus development capacities on areas that do not necessarily match the products that are actually in demand.

[102] For a comparison with the incentives in the field of science, see LEE, MOISA AND WEISS (2003), p. 23. In this area, scientific publications serve to build up reputations in order to signal skills and hence improve career opportunities as well.

[103] Cf. MENDYS-KAMPHORST (2002), p. 20. Empirical data seem to support this theory: "Contributing to open source did help many programmers to get access to venture capital or to be offered attractive jobs by commercial software developers." IBID (2003), p. 12.

[104] LERNER AND TIROLE (2002), pp. 15-16 point out that similar trends can be found in scientific research as well. While some fields are researched intensely, other areas remain completely untouched.

[105] Cf. WEBER (2000), p. 22.

The practical design of open-source projects has indicators pointing out the importance of signaling:[106]

- Stating the development contribution is of particular importance in a project.[107] All developers involved are listed in the project history, the credits or the maintainer list. There are also many web pages of open-source projects where the involved developers list the significance of their contributions. For example, the web page of the Apache web server very clearly shows the involved developers and their respective contributions.[108] The Sourceforge web page also ranks the registered projects and the developers along with their qualifications and contributions.[109] Furthermore, it is considered a major offense if the contribution made by a developer can no longer be found. "*Removing a person's name from a project history, credits or maintainer list is absolutely not done without the person's explicit consent. … surreptitiously filing someone's name off a project is, in cultural context, one of the ultimate crimes*."[110]

- The first open-source projects in particular were technically challenging solutions for operating systems. These solutions are best suited for establishing a reputation and therefore offer a higher signaling incentive than tasks that benefit less advanced users. [111]

- The modular structure of many projects does not only facilitate distributed teamwork in different developer teams, it also makes it possible to allocate the individual contributions more exactly.[112]

This examination of motives has shown that there are also numerous economic explanations for a developer's participation in an open-source project, as developers can also use their participation to improve their career prospects. It is imperative in this case that a proprietary software market exist. If a developer pursues his or her own interests, that will not necessarily lead to the best result for the software user. As has been shown, developer interests are best met when

[106] Cf. SCHMIDT AND SCHNITZER (2003), p. 11.
[107] Cf. MENDYS-KAMPHORST (2002), p. 13.
[108] Cf. www.apache.org
[109] Cf. sourceforge.net
[110] Cf. RAYMOND (1998b)
[111] Cf. WEBER (2000), pp. 21 et seq.
[112] Cf. LERNER AND TIROLE (2000), p. 17.

technically demanding solutions are designed. These demands, however, are generally different from those that end users place on software.[113]

2.3.3 The Development of Open-Source Software - Not A Bazaar

In one of the best-known articles on the principles of open-source development, the work in an open-source project is likened to a "bazaar": "The Linux community seems to resemble a great babbling bazaar of differing agendas and approaches… out of which a coherent and stable system could seemingly emerge only by a succession of miracles."[114] The development work in a proprietary project, on the other hand, is viewed as the "construction of a cathedral" and thus as a strictly hierarchical form of organization.

Though the example of the bazaar clearly demonstrates the multifaceted nature of open-source cooperation, an open-source project lacks the very attribute that defines a bazaar: the coming together of suppliers and demanders and the resulting establishment of a price for the traded good. A bazaar is the prototype of a market.[115] The seemingly chaotic trading on a market is the manifestation of a decentralized coordination mechanism that brings together the wants of the demanders and the opportunities of the suppliers. The bazaar serves as a basis for information exchange and pricing. Prices indicate relative shortages and provide important information to suppliers about what value demanders place on a good and which goods are in highest demand. At a bazaar, every dealer pursues his or her own interests by maximizing his or her profits. This can be best achieved if the vendor offers goods that are demanded especially urgently by customers. The vendor who best satisfies the customers' wants stands to gain the most. As such, the demander benefits indirectly from the self-serving interests of the suppliers.[116]

In an open-source project, suppliers and demanders do not meet. The open-source bazaar only consists of suppliers and developers who develop their software primarily according to their own ideas. Although developers, as the previous section showed, also pursue their own interests, there is no connection to the interests of

[113] See also JOHNSON (2001) regarding a model analysis of the development of open-source software and of developer contributions.

[114] Cf. Raymond "The cathedral and the bazaar."

[115] See BORCHERT, GROSSEKETTLER (1985), pp. 13 et seq., for the constitutive role of pricing for a market.

[116] Cf. SMITH (1776), book 1, chapter 2, who pointed out this mechanism more than 200 years ago: "It is not from the benevolence of the butcher, the brewer, or the baker that we expect our dinner, but from their regard to their own interest. We address ourselves, not to their humanity but to their self-love, and never talk to them of our own necessities but of their advantages."

demanders and users of a software. The software that is created in the pursuit of the developers' interests is not necessarily the software demanded by the customers.

2.3.4 The Role of the Market and Fulfilling the Market Functions in Open Source

In a market economy, supply and demand are coordinated by the market through competition between different suppliers. In contrast to a planned economy, the market is a decentralized planning and coordination tool, i.e., there are no national economic targets with respect to output or rationing nor are there regulations concerning the quantities to be demanded. As shown in the previous section, the suppliers' self-serving interests are what motivates them to optimally meet the demanders' interests.

If shortages exist in a market, a rationing method must be found that regulates the way demanders compete for scarce goods. However, shortages can also be present in the end-user market when production factors are used. There are many possible behaviors and regulations with which demanders organize the distribution of scarce goods and influence their own supply of these goods: in this regard, waiting in lines, using or threatening the use of violence, bribery and fraud can be competitive methods in the market that are just as legitimate as competing through price and quality. However, these methods operate in extremely different ways. Accordingly, one cannot choose between living in a society with or living in one without competition, as noncompetitive societies cannot exist in a world of scarcity. What can be chosen, however, are the competitive methods, i.e., the rules governing how competition is carried out.[117]

The decision to utilize the market as a coordination tool also entails a decision to live in a decentralized economy that grants the participants far-reaching freedoms of planning, action and choice. Markets are not an end in themselves but serve to fulfill various functions that are summarized in Table 4:

- The principle of **customer sovereignty** requires that supply and demand are balanced according to the needs of the demanders. Individual plans are balanced out through the price mechanism in and between the individual markets. Adequate price variations reduce excessive supply (waste) and excess demand

[117] Cf. VANBERG (1997), pp. 707 et seq.

(shortage) and as a result balance out the individual economic plans of market participants. In this context, suppliers have to let themselves be guided by the demanders and not vice versa: the final decision about the success of a supplier is made by the demanders through the purchase of a good.

- The **factor allocation function** steers scarce resources to their most productive use. Assuming that there are scarce means of production but unlimited demand, the scarce means must be steered so that they are used in a way in which they are needed most urgently.
- The **distribution function** of competition distributes income in accordance with the production contribution. Consequently, the one who better meets the needs of the demanders receives a higher compensation than others.
- The **adjustment function** ensures that the behavior of market participants quickly conforms to new conditions. If, for example, a supplier can no longer survive in the market with his or her product because the demanders' needs have changed, the offering of this supplier will either have to be adjusted accordingly or withdrawn from the market.
- The **progress function** provides new incentives to develop new products and processes and consequently to provide better cost-benefit combinations than the previous ones (incentive function of competition).

Table 4: Market Functions and the Consequences of their Absence in Open-Source Software

Market function	Ensures	Its absence in open-source software leads to:
Customer sovereignty	Balancing of supply and demand	Under-supply Over-supply
Factor allocation	Scarce resources are steered towards the most urgent need	Misallocation of resources
Distribution	Income distribution by production contribution	Nonsustainability
Adjustment	Structural adjustment	---
Progress	New products New procedures	Innovation obstacles

.

The price system is of key importance within the competitive coordination and planning process: for a decentralized market economy, in which numerous autonomous decision makers must interact in a coordinated fashion, prices constitute the central navigation system for all parties involved. This is made clear in the different functional aspects of a price:

- **Information**: The price reflects shortages and is therefore, on the one hand, an indicator of urgent demand (willigness to pay) and, on the other, of the consumption of resources associated with the use of goods (cost information).
- **Steering**: Changes in one or both of the market sides lead to price changes and induce quantity reactions both in the observed market as well as in any upstream and downstream markets.
- **Motivation**: The price determines the seller’s income: the better customer wants are satisfied, the higher the income. This results in a permanent incentive to innovate.
- **Assessment**: The success of an economic activity can only be assessed and compared with other activities by means of prices.

Developers lack key information due to the absence of pricing in open-source software. They do not have information concerning customers’ willingness to pay (= actual preferences), based on which production decisions would be made in the market process. Because of the absence of this information, supply does not automatically develop in line with the needs of the users, which may manifest itself as oversupply (excessive supply) or undersupply (excessive demand). Furthermore, the functional deficits in the software market also work their way up to the upstream factor markets (in particular, the labor market for developers) and – depending on the financing model of the open-source software development – to the downstream or parallel complementary markets (e.g., service markets) as well.

Because the open-source model at its core deliberately rejects the use of the market as a coordination mechanism and prevents the formation of price information, the above market functions cannot be satisfied by the open-source model. This results in a systematic disadvantage in the provision of software in the open-source model as compared to the proprietary production process. Aside from the market functions and their tasks, Table 4 also shows in note form the consequences of their absence in open source. The following points deal with the individual aspects in more detail. The limits of open source that are a direct consequence of the absence of the market as a coordination tool are discussed in more detail in the following points:

- Developer orientation is not customer orientation
- Less effective utilization of resources
- Lower innovation capacity

The following section also discusses the less favorable development of standards that result from the software-specific attributes.

2.3.5 Limits of Open Source

2.3.5.1 "Happy Engineering" - Developer Orientation Instead of Customer Orientation

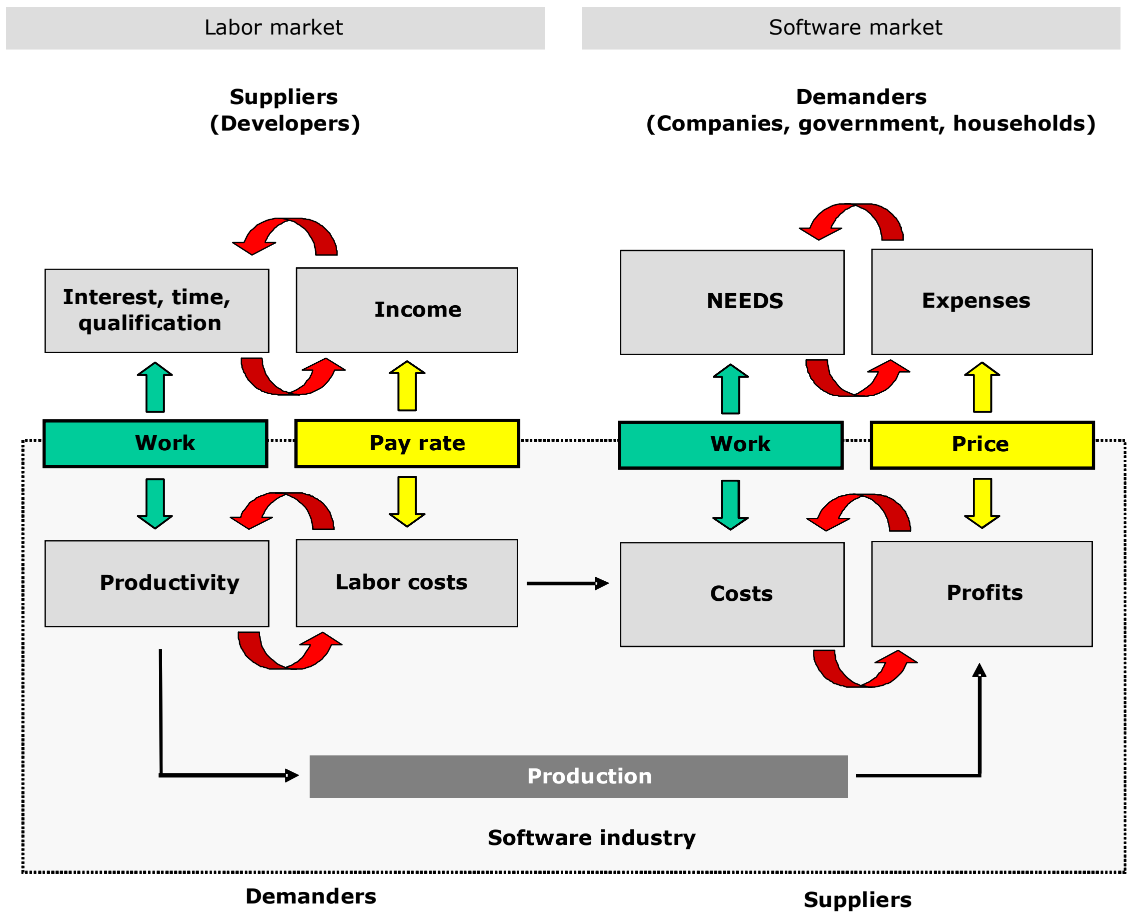
In evaluating a new product, the key criterion is not the technically feasible maximum; in the final analysis, it is only the marketability of a development. If, for example, engineers focus solely on the technical aspect in their developments, they will create expensive and sophisticated high-performance products that may not find a buyer because the end user cannot recognize the added value or is unable to afford it. "Happy Engineering" is the term for this type of development, one that exhausts the technical possibilities without considering usability, operability, added value and incurred costs. A determining factor in the distribution of products is not their technological sophistication but the creation of additional value for the customer.[118] In the case of proprietary software, the market matches the technologically feasible solutions with the wants of customers and their resultant willingness to pay. In the case of open-source software, the absence of pricing prevents matching from occuring (cf. Illustration 9 and Illustration 10 regarding the following section).

Proprietary software can only survive in the market if it optimally fulfills customers' wants.[119] This does not mean, however, that the best software necessarily has to be the technologically most sophisticated solution. For a business, the value of a product depends on the utility that the customer derives

[118] Cf. BACKHAUS (1999), pp. 130-133, where the term "Happy Engineering" is also mentioned.

from the product. This customer valuation is reflected in the price that the business can demand. The greater the added value for the customer, the greater the company's profits. Therefore, the identification of user needs and the resultant willingness to pay make up the first step in traditional software development. The more valuable the software is to demanders (quality or universality), the higher the price/sales volume and as such the profits that a software developer can expect for his or her software.

Illustration 9: Market System in Proprietary Software Production

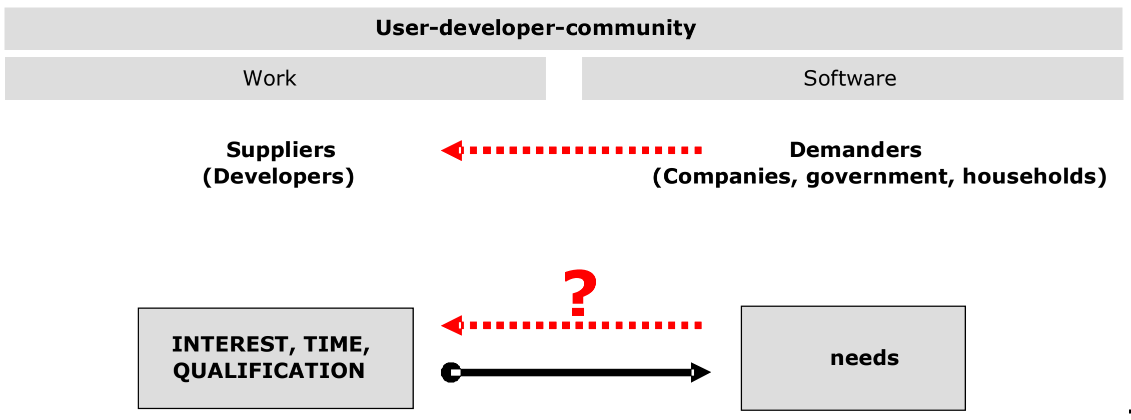


In market production, realizable profits are matched to the production cost (shown in Illustration 9 in simplified form as labor costs). The production of a software product is only economically justifiable if the demanders are willing to actually finance the costs incurred in software production through the product price.

[119] This includes user friendliness (in particular in the case of mass markets) and compatibility with previously installed platforms (investment protection, use of network effects).

Otherwise, the scarce resources used in production should rather be steered towards other, more productive uses. Since proprietary software developers maximize their profits by developing software in line with the customers' needs, there is a strong incentive for them to invest in market research and to actually identify the customers' needs.[120]

Illustration 10: Open-Source Software Production


User-developer-community
Work
Software
Suppliers (Developers)
Demanders (Companies, government, households)
?
INTEREST, TIME, QUALIFICATION
needs

Open-source software development, on the other hand, does not consider the needs of a broad user base. For that reason, the needs of the users do not play a significant part. This is not deliberate but due to a lack of information about the actual preferences of demanders. The market system degenerates, becoming a lopsided relationship between developers and recipients. The above examination of the motives for participating in an open-source project showed that developers develop primarily for their own problems or needs. As a result, however, the needs of nondevelopers are difficult to pinpoint and implement. If not all the users are developers, this may lead to software being developed that does not meet the demands of the users (i.e., the customer, the market).

As a result of the signaling incentive discussed above, a programmer is interested more in developing technically complex software than a technically less sophisticated application, as the latter would entail less of a reputation gain. An open-source developer therefore has very little incentive to identify and solve problems of less knowledgeable users.[121]

[120] Cf. SCHMIDT AND SCHNITZER (2003), p. 14.

[121] Cf. HANG AND HOHENSOHN (2003), p. 44 and EVANS AND REDDY (2002), p. 30.

As a result, customer wants are only considered in the development of open-source software if the users themselves become developers. This corresponds to the original cooperative idea behind an open-source community. It also means, however, that processes based on division of labor must be restrained, gains from specialization must be relinquished, and as such the major driving force behind productivity progress and economic growth must be weakened.

The example of the Apache web server shows that open-source software can currently be extremely successful if users are technologically adept. In this case, users benefit from the adaptability of the software. Because they have the appropriate technical skills, documentation and easy handling are not the main criteria in their software decision.[122]

Companies involved in business models based on open source have a greater interest in taking customers' wants into consideration. However, it should also be noted that the developed software must be freely accessible, even in commercial business models. This means, though, that no company is able to gain a specific competitive advantage by developing highly user-friendly software. The additional open-source services are more successful if the additional service can only be marketed by the offering company. The business model of distributors can be interpreted to the effect that their distributions attempt to add customer utility to open-source software that open-source development alone cannot add. However, the possibilities for exclusively marketing the resulting product are very limited in this regard as well.

2.3.5.2 Inadequate Allocation of Resources

Due to a lack of price mechanism, the open-source model does not provide any information about the value a user places on a software product. The advantage of a price system is that different goods and services can be evaluated based on a common standard and are thus comparable.[123] In the open-source model, it is not possible to compare the urgency of alternative needs. As shown below, this valuation deficit immediately works its way up to the upstream factor market, where not only does the open-source model not allow income to be produced, but it does not permit any shortage signals to be generated.

[122] Cf. Lerner and Tirole (2000), pp. 8-9 and Schmidt and Schnitzer (2003), p. 14.

[123] Despite the many claims of "not being able to compares apples with pears", the price mechanism allows exactly that comparison.

If open-source development is considered a leisure activity, every developer is free to individually design his or her time plan that is not subject to any economic evaluation criteria except the developer’s. If, however, the open-source development method is viewed as a basic alternative to the proprietary model, it has to be compared to the proprietary model in terms of how it allocates resources. Only then can one judge whether it is capable of producing the desired products with the least-possible effort. The shortages emerging in open-source development are not related to the allocation mechanism for created products. In this connection, there cannot be any shortages because there is nonrivalry in consumption. In the open-source model, shortages come about instead because limited development capacity has been allocated to a proper use. If the open-source model is to be viewed as a general alternative to the proprietary world, such shortages are inevitable.

Even if open-source software is distributed free of consideration, the development of software still has its costs, economically speaking. Developers are confronted with opportunity costs, as they can use the time available to them but once. Without a price signal, it remains unclear whether development time spent in Project A would create greater utility if spent in Project B. The traditional economic view of software, in which it is a nonrival good among users, therefore only applies to existing software, that is, from an (economically less interesting) ex post perspective. However, whenever the issue is about using scarce resources for the production of new software, competition among potential future users definitely exists if they are faced with the choice of having to do without the new Software A so that the alternative Software B can be programmed, or vice versa (ex ante rivalry). Markets can easily solve this conflict through the price mechanism, while other coordination processes fail in this aspect as they are unable to valuate and consequently compare A and B due to the lack of pricing.

Therefore, from an economic perspective, the lack of information about the actual wants of demanders leads to a poor allocation of resources, where developer capacity is the main resource. In a price-based software market, producers would be able to focus their development work on those products that are needed most urgently. The open-source model is not able to identify the desired demand in such a fashion. A lot of developer capacity may thereby be tied up in products that are not needed or that would also yield good results with less development work.[124] The signaling goals of developers merely serve to exacerbate this tendency: it is likely that areas promising a higher profile are occupied by many developers, while hardly any development work is performed in other areas. Because there is no opportunity to allocate resources to their proper use, the open-source model is not suitable as a basic production method for software.

Consequently, the open-source development method must consequently be classified as neither economically efficient nor effective.[125] It is less effective than the proprietary model as there is no mechanism to check whether developer capacity is actually invested properly. It is not efficient because the input can be unreasonably high compared to the output.

The allocation of resources can be improved if more commercial elements are integrated in open-source projects. Companies will invest in those open-source projects that they suspect will provide the greatest additional benefit for their business model. However, in this case the resources are allocated only indirectly by the utility that the complementary product or service offers to the providing company. The following section examines how the economic coordination efficiency of such complementary strategies can be assessed.

2.3.5.3 Sustainability of Complementary Open-Source Strategies

When it comes to financing the development costs incurred in developing the nonmarket open-source core (cf. Illustration 7), many people refer to open-source business models based on open-source core products (complementary strategy). This amounts to a cross-subsidization of open-source software production from the profits generated in downstream or parallel value-added steps. Depending on the contestability of the complementary markets, there are two different scenarios in which this strategy is assessed:

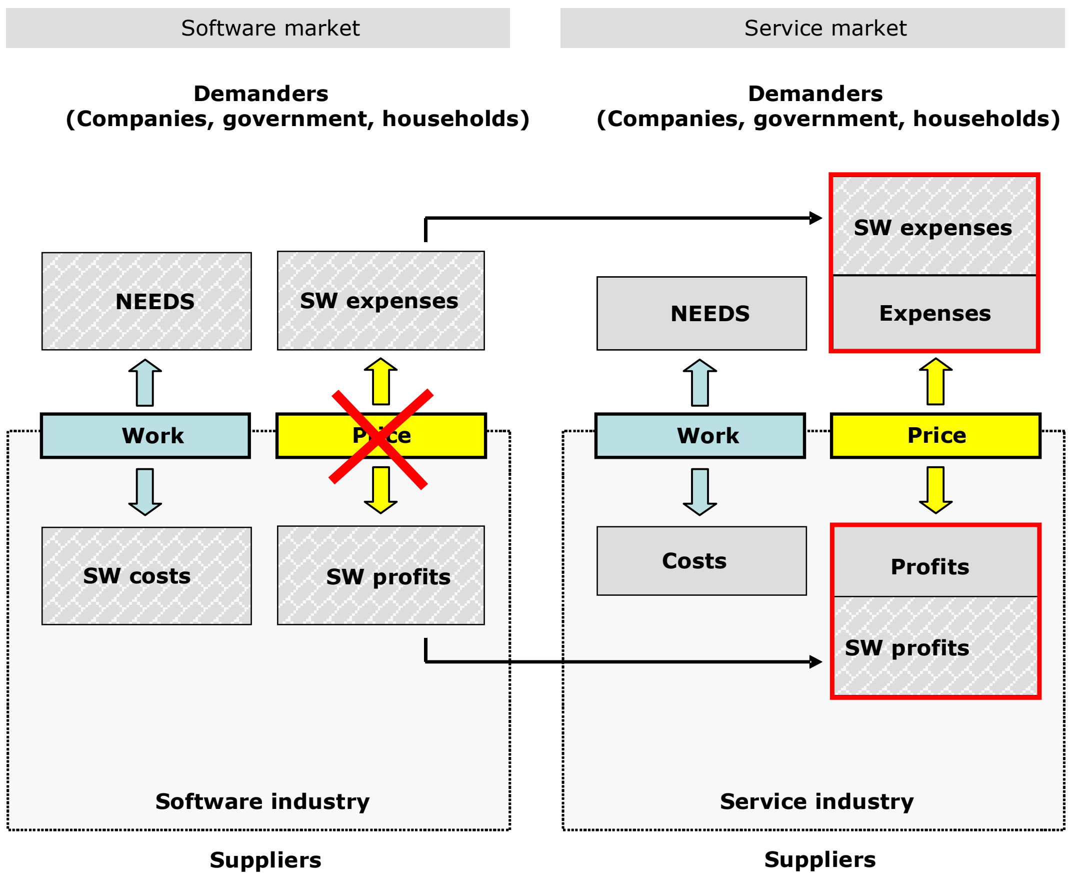
- Scenario 1: Cross-subsidization possible (no contestability)
- Scenario 2: Cross-subsidization fails (contestability)

[124] Though such products could also come into existence in the proprietary software market, bad or inappropriate products are sanctioned as lossmakers by that market.

[125] Efficiency is the ratio of output to input ("doing things right"). Effectiveness is the ratio of targeted utility to actual utility ("doing the right things").

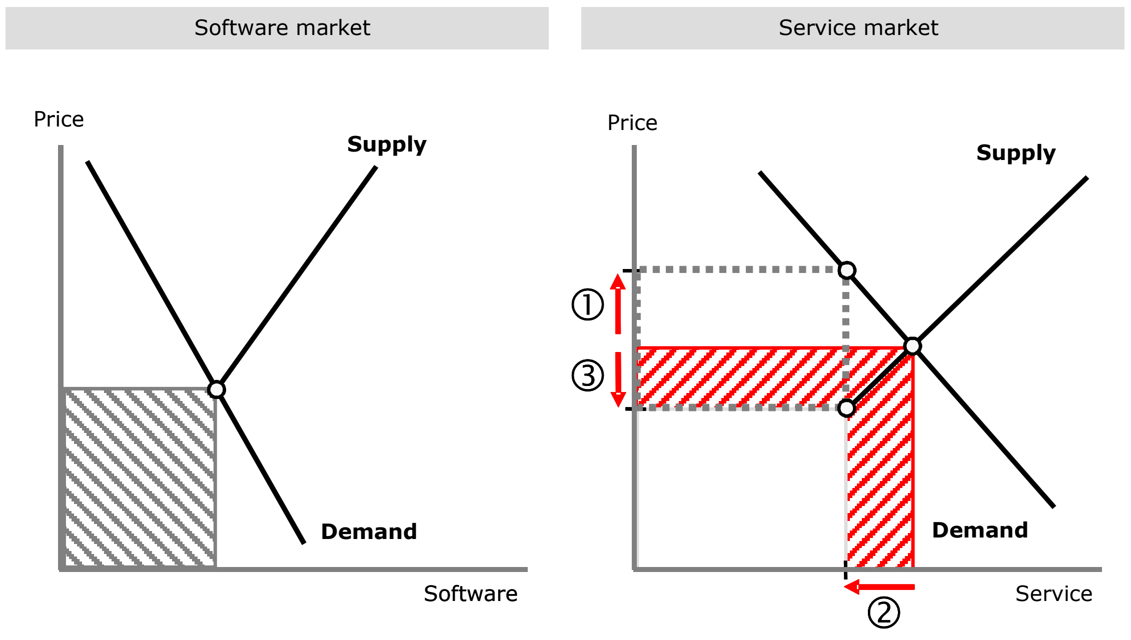
Scenario 1 is examined in more detail first. Illustration 11 shows the basic construction of this financing model in the software market (open-source core) and in the service market which acts as a commercial complementary market based on that software market.

Illustration 11: Complementary Strategy with Cross-Subsidization



As there is no pricing in the software market, an attempt is made to recover incurred costs by selling software services at prices higher than cost. Starting with an original market equilibrium (in the case of a proprietary organization of both markets), the resulting price increase in the service market will trigger certain adjustments, as the demanders in that market are not willing to buy the previous equilibrium quantity for a price that is clearly above the previous equilibrium price. The market’s price and quantity reactions to the nonpricing of software and its cross-subsidization by a complementary market are shown in Illustration 12.

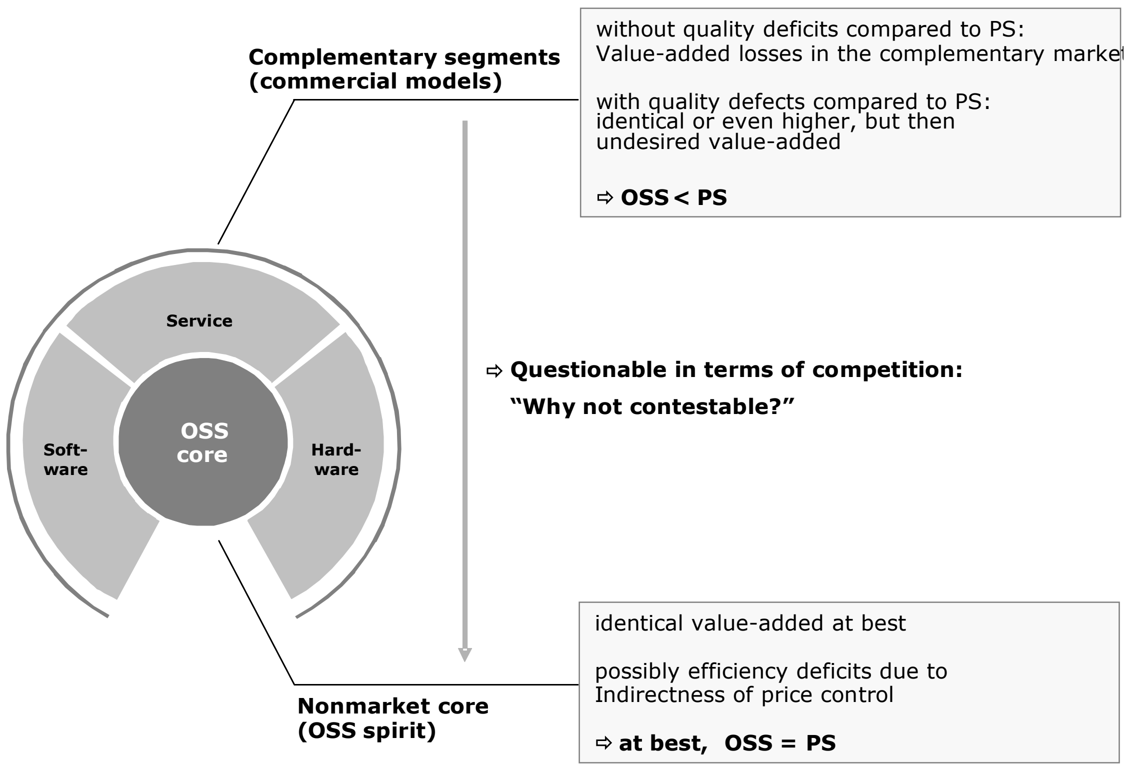
Illustration 12: Adjustments Triggered in the Complementary Market



The shaded area in the diagram on the left corresponds to the lost value-added in the software market caused by nonpricing. If that loss is to be recovered by a corresponding price increase in the service market (dotted rectangle in the diagram on the right), it will lead to a gross price increase in the complementary market (effect ①), which causes the demand/sales quantity to be reduced (effect ②). At the same time, this reduces the net price for services (effect ③) and thus diminishes value-added in the complementary market, which corresponds to the shaded area in the diagram on the right. The overall result is one of deadweight losses similar to those seen in the analyses of tax effects. In both scenarios, the deadweight loss is caused by prices for demanders and suppliers moving further apart. Cross-subsidization has the effect of a wedge, driving purchase and selling prices apart and reducing economic welfare by triggering quantity reactions.

The discussions so far have been based on the assumption that an open-source software product entails the same services as proprietary software, i.e., that both production processes lead to identical quality software. The net loss in value-added caused by the complementary strategy in the entire IT sector could be prevented only if the service demand for open-source software were greater than for proprietary software. Considering that service demand and product quality are in a reciprocal relationship, that means that the loss in value-added in open-source software production could only be countered by worsening software quality (higher training costs, more support etc.). However, this form of artificial value creation can be classified as economically undesirable. In this case, it would be better to permit added value to be created in software development in order to provide the economy with better-quality software. Otherwise – as has been shown – customer sovereignity will not be sufficiently asserted.

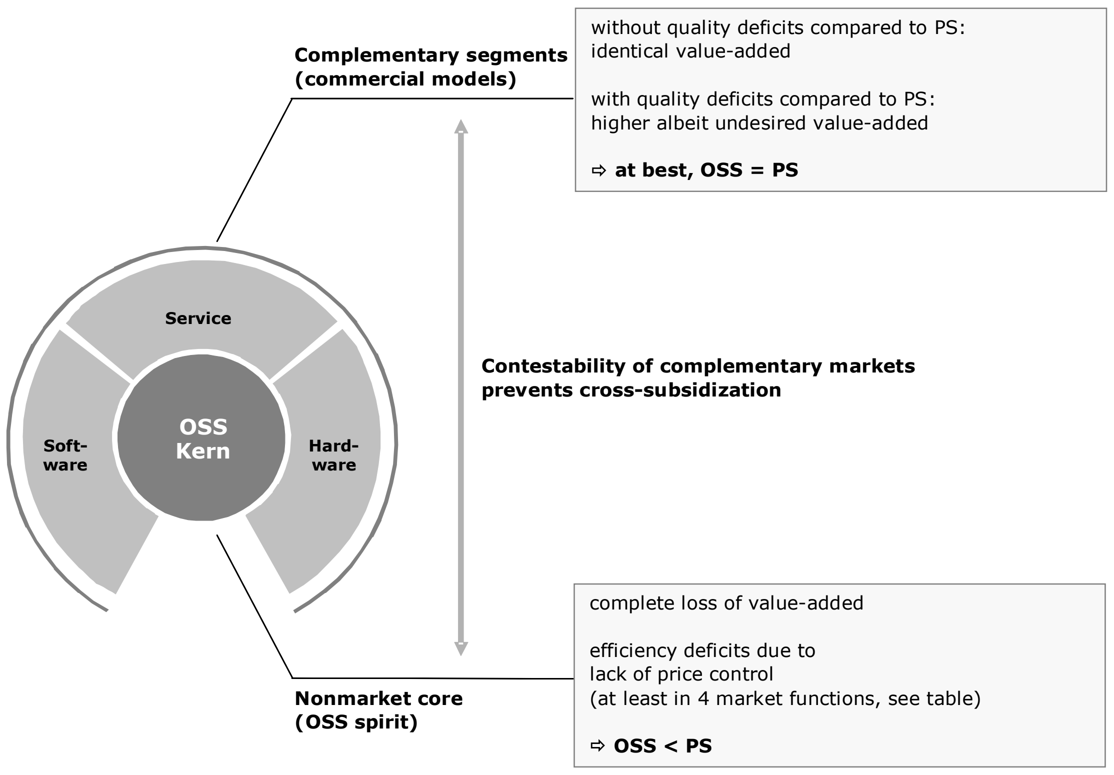
Illustration 13: Assessment of Complementary Strategy with Cross-Subsidization


Complementary segments (commercial models)
without quality deficits compared to PS:
Value-added losses in the complementary market
with quality defects compared to PS:
identical or even higher, but then undesired value-added
⇨ OSS < PS
Service
Soft-ware
OSS core
Hard-ware
⇨ Questionable in terms of competition:
"Why not contestable?"
identical value-added at best
possibly efficiency deficits due to Indirectness of price control
⇨ at best, OSS = PS
Nonmarket core (OSS spirit)

In wrapping up Scenario 1 and moving along to Scenario 2, one might wonder how cross-subsidization can be successful at all, as the complementary markets are basically contestable (if not, there is a serious competition problem). Since open-source software is open to anyone even without a financing contribution, service providers might, for example, appear in the market offering their services at cost, thereby forcing out those providers who have to set selling prices that will allow them to recover the financing contributions for open-source software.

If the complementary markets are contestable (Scenario 2), the cross-subsidization strategy has no foundation. This leads to a loss of value-added in the IT sector in an amount equal to the shaded area in the diagram on the left in Illustration 12. As there is no indirect connection to commercial business models, the coordination deficits in open-source software production stated in Table 4 reach their full effect. If the loss in value-added in software production is compensated for by higher "value-added" in the complementary sector, this will require worse product quality, which in turn would mean a failure to reach the goal of customer sovereignty.

Illustration 14: Assessment of Complementary Strategy without Cross-Subsidization


Complementary segments
(commercial models)
without quality deficits compared to PS:
identical value-added
with quality deficits compared to PS:
higher albeit undesired value-added
⇨ at best, OSS = PS
Service
Soft-ware
OSS Kern
Hard-ware
Contestability of complementary markets prevents cross-subsidization
Nonmarket core
(OSS spirit)
complete loss of value-added
efficiency deficits due to
lack of price control
(at least in 4 market functions, see table)
⇨ OSS < PS

If one traces the different branches of these scenarios to their logical conclusions (with/without cross-subsidization, with/without quality reduction), it becomes clear that, in the end, the results in the open-source model are clearly inferior to those of proprietary software production. This overall result is ultimately attributable to the absence of pricing for open-source core products.

2.3.5.4 Lower Innovation Capacity

Innovation means that new knowledge is used to offer products or services that the customer prefers to previous solutions. Therefore, an innovation is the commercialization of a product that was not previously commercialized.[126] In this

[126] Cf. Iansiti and Lerner (2002), p. 2.

context, innovation is understood in terms of adoption as well, i.e., the consumers adopt a new product or process. By contrast, an invention is the *creation* of a new product or process. Therefore, in order to measure innovation, one must analyze its acceptance or its influence on the private consumer, on organizations and on society as a whole. For that reason, information concerning the distribution of a product also provides information on innovativeness.[127]

Innovations are extremely important to economies. They are customarily subdivided into product innovations and process innovations. In the case of a product innovation, a new process leads to the production of new products or product qualities. Process innovations, on the other hand, allow for an elimination of production factors and consequently for a reduction in production costs.[128]

In static markets[129], the measure of economic well-being is the sum of consumer surplus and producer surplus. This sum is also called social surplus, which is maximized when a good is priced at marginal costs. From a static point of view, the pricing of open-source software at marginal costs is efficient. However, the software market is a dynamic market characterized by rapid technological changes. Pricing at marginal costs in this case does not offer companies enough incentives to invest in software development.[130]

The existing incentives for professional software developers to fix bugs or customize software causes personal utility to increase for developers. However, the incentives for software innovations in the proprietary market cause utility to increase for the economy.

Innovations always represent a critical success factor for companies, because only if they can produce more successful innovations than their competitors will they prevail on the market and generate enough returns to cover their costs. Companies will invest in the development of software if they can protect the intellectual property rights in the software created and if, in compensation for possible losses in the case of failure, they can achieve above-average prices for the product.[131]

[127] Cf. IANSITI AND LERNER (2002), p. 3. See the examination by LERNER AND TIROLE (2002) for more information on the interrelationships between licensing and innovations.

[128] Cf. FRITSCH, WEIN AND EWERS (2003), p. 75.

[129] Static markets contain a given technology exists and do not produce innovations.

[130] Cf. SCHMIDT AND SCHNITZER (2003), pp. 6 et seq.

[131] Cf. SCHMIDT AND SCHNITZER (2003), p. 8.

For that reason, a company only has an incentive to undertake innovations wherever they help the company to realize first-mover or pioneering advantages that reward it for its expenses, particularly for the risk associated with the innovation activities. The pioneering advantage is the return resulting from the fact that no other company can claim the innovation. The longer it takes for its competitors to adapt in response, the greater the pioneering advantage of the innovator and the greater the incentive to innovate. If, however, competitors are in a position to immediately imitate the innovation, or, as is the case with open-source software, if the company must make it immediately available, pioneering advantages cannot be realized. An extremely quick reaction on the part of competitors upon the launch of new products or processes would, for the overall economy, result in a reduction or the complete elimination of incentives to innovate.[132]

The prospect of future profits is thus the strongest incentive to undertake innovations. These incentives are not present in the open-source model, as the disclosure of the source code provides every user with the ability to customized the software as needed. The software can be freely copied and distributed as well. Companies building their business on the open-source model have therefore much less incentive to invest in R&D activities, and it is doubtful they will be as innovative as a supplier of proprietary software. The only incentives for a company to invest in the enhancement of software are indirect, if at all, because in the open-source world, as has been shown, companies are not able to generate profit with the software itself but only with additional services or products.

For example, one can envision two companies identical in all respects except that one invests in open-source software development and the other does not. The first company has to make every software improvement it develops (especially under GPL) available directly to the second company. As such, the open-source development process does create equal business opportunities for both companies, but the first company will, because of its development work, inevitably have higher costs than the second company. The first company can be successful only if the open-source development fosters added trust among customers or if it has shorter development times for proprietary software or services that are based on open-source software. [133]

[132] Cf. FRITSCH, WEIN AND EWERS (2003), p. 77.

[133] Cf. EVANS AND REDDY (2002), p. 30.

As a result, a basic advantage of proprietary software is the fact that it offers developers an opportunity to recover the investments they have made and to transform the additional utility they created for consumers at least partially into earnings for themselves. *"Like in all other industries, the profit motive provides a very powerful incentive to innovate that is not present in the open source world".*[134].

Illustration 15: From an Idea to Marketability

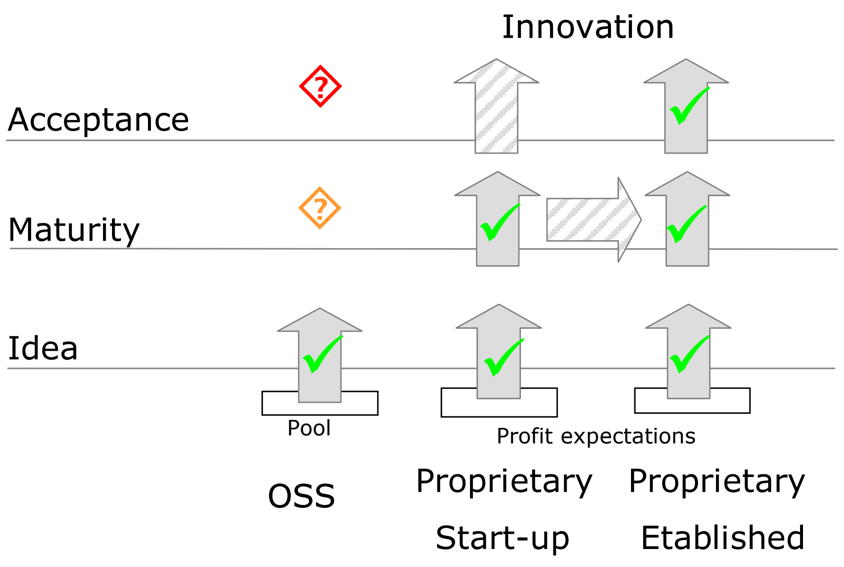


Illustration 15 is a simplified model comparing the different paths that lead to an innovation. For an innovation to come about, there must first be an invention or idea. In the open-source model, it develops out of the large pool of developers. In this context, as has been shown, developers focus on their own interests or problems. Because of its free form of cooperation and the huge number of developers, the open-source model is able to produce a multitude of ideas (brainstorming effect). However, these are not selected for their exploitability. Proprietary software producers, on the other hand, focus on the customers' interests, for they have to expect that there are enough takers for their product for their investments in the software development to pay off. In this context, the proprietary development model differentiates between start-ups and established companies. Principally, the initial conditions are identical for both companies. Numerous software developments have been started as small start-ups in the past and garnered considerable shares of the market with their product.

[134] See CF. SCHMIDT AND SCHNITZER (2003), p. 13.

The maturing process in open-source software is expedited more or less depending on how many developers get together to further develop a project. From this stage on, proprietary start-up companies may also be taken over by an established company. The incentive scenario remains basically the same. The established company will be interested in an acquisition only if the cost-benefit ratio is deemed particularly favorable by its demanders.

The last stage finally is the market penetration and wide distribution of the product. An innovation is not relevant merely as an invention of a new product or production process. Rather, an innovation has an economic impact only if it is widely distributed and provides the users with additional value in the form of a productivity boost or lower costs. If software development is not aligned with the users' preferences, however, it will not bring forth innovations that are widely distributed and as such work to increase well-being in an economy.

Open-source software must also be comparable to proprietary software with regard to innovativeness. However, the strengths of open-source software used to lie mainly in copying existing software. Even the start of open-source software had the goal of copying software, that is, the Unix operating system. "*Clearly, much innovation in commercial software has occurred over those 25 years. Just as clearly, much (but certainly not all) of the focus of GPL software over the past two decades has been on creating "free" versions of proprietary software.*"[135]

2.3.5.5 Standards are More Difficult to Establish

As shown in the first section, standards[136] play an important role in the software market. They make it possible to fully utilize network effects on the supply and demand sides. Suppliers developing software for a given standard benefit from it in that they do not have to adapt their products to different platforms. Demanders benefit from a standard that, for example, eases file sharing for them and obviates the need to be trained on differing platforms at different workstations. Another consequence of standardization is that there are pre-defined interfaces between programs developed by different suppliers. This ensures that program functionalities like copy and paste can be used universally. In the open-source model, the formation of standards can be more difficult than in the proprietary

[135] Cf. EVANS AND REDDY (2002), p. 49.

world as there are no economic incentives to establish and hold on to a certain standard. The problem of network fragmentation and thus the nonestablishment of a standard can occur on the development and the distribution / producer levels.

If this happens already on the development level, it is called "forking". Developers may not see eye to eye concerning the direction that a project should take, which might lead to a breakdown into different projects that are further developed independently of each other and that are not intercompatible.[137] The Unix operating system is the best-known example for the emergence of many different and partially incompatible versions.[138] This was exacerbated by the fact that commercial suppliers put their own Unix version into circulation in order to set themselves off from their respective competitors.

It is true that forking can be understood as a type of competition between different development trends for the best solution. Once again, however, the emphasis is on developing the technologically best solution. The final consumer is, however, equally interested in practical issues such as compatibility or interchangeability of documents. Because proprietary suppliers have a vested interest in distributing their software as widely as possible, they will try to achieve equally wide coverage with their products. Furthermore, the presence of a certain degree of market power is helpful in establishing a standard or platform.

The standard can also become fragmented on the level of distributors and hardware/software producers whose business models are associated with open source. There are many distributors trying to market their own Linux distributions. This has resulted in a slew of Linux versions that are not necessarily compatible with each other.[139] For example, distributors can choose different approaches for organizing file storage. This makes it difficult for software application developers to develop a routine for setting up applications as it will not run under different versions. Developers must therefore offer different program versions for different distributors, or these programs must be adapted by the distributors.[140] In the same

[136] "Standards" or "standardization" are defined in the following as measures to prevent forking effects. The difficulties that may arise in the development of open IT standards such as TCP/IP are not discussed here.

[137] Cf. MENDYS-KAMPHORST (2002), p. 15 and p. 39.

[138] GRÖHN (1999), p. 11, again methaphorically, refers to a "Balkanization" of Unix.

[139] For an overview of the different Linux versions, see, for example, www.suse.de/de/index.html oder www.redhat.com/

[140] Cf. MENDYS-KAMPHORST (2002), p. 48.

way, programs that are operable on one user interface do not have to run the same way on another user interface.[141]

The special importance of standards in the software market and the need to establish them in the open-source world is evidenced by the creation of different initiatives that attempt to set uniform rules for file storage and other areas.

2.4 Future of Open-Source Development and Economic Implications

2.4.1 Proprietary and Open-Source Software - A Comparison

The starting point of the preceding analysis was the absence of a market as a coordination tool between software developers or suppliers and software users.

Table 5: Comparison of Proprietary and Open-Source Software

	Proprietary Software	**Open-Source Software**
Supplier-demander coordination	Market as coordination tool	Autonomous software engineering
Development orientation	Orientation towards customer needs	Orientation towards developer interests
Innovation incentives	Pursuit of profits by innovator Ownership of innovations	Personal interests that do not necessarily match user interests
Compatibility and interoperability	Strong incentives for compatibility	Threat of forking
Bugfixing	Easy installation, high level of compatibility in different hardware constellations, but because of that longer process	Quick availability, but user-friendly installation and compatibility may be difficult
Customizability	Customizability within the defined possibilities	Far-reaching customizability possibilities for experienced users
Signaling	Depends on the producer's publication policy	Strong signaling effect due to disclosure of individual developer contribution

Because open-source development consciously shuns market processes, it does not carry out important economic coordination tasks that, in proprietary software production, are performed by the market. The previous section pointed out the

[141] EVANS AND REDDY (2002), p. 44 produce this example for the KDE and GNOME interfaces.

resulting consequences for various market functions. Table 5 contrasts the most important statements on proprietary and open-source software once again.

The differences between proprietary and open-source software result in the respective software models having different strengths as well. The **strengths of proprietary software** reside in:

1. Wellspring of Innovation

Owing to the granting of ownership rights, there are strong incentives in the proprietary market to invest in research and development and to create innovations.[142] In the open-source model, on the other hand, all development results are open to all other parties involved. If an innovation does not consist of many small changes but of fundamental development trends, proprietary software development will offer organizational structures that are better capable of realizing market maturity.

2. User or Customer Orientation

The engineering of proprietary software is customer-oriented; this is contrasted by a developer orientation in open-source software. Open-source developers are often guided by technical and not customer-oriented considerations (such as user friendliness and operability). While the market mechanism ensures that proprietary software is developed according to the wants of the customers, there is no incentive mechanism in open source to steer the interests of the developers towards the wants of the customers.

3. User Friendliness: High Degree of Usability and Ease of Use

In standard applications for the mass market, proprietary software development has to be guided by the widest-possible customer base. In order for the products to cover a large portion of the market, they should be accessible to less advanced users as well. This means that comprehensive software documentation should be provided – a characteristic that is often lacking in open-source software. The quality of a proprietary product and hence the success of a company are decided by the sales figures and consequently by customer satisfaction.[143]

[142] Cf. SMITH (2002), p. 77.

[143] On user friendliness, see HORST (2003), p. 36.

4. High Level of Standardization, Compatibility and Updates

Proprietary software suppliers have a vested interest in their products being highly standardized. For that reason, compatibility is realized in different hardware environments. Hardware and software producers can safely assume that their products will always work the same way with defined platforms. This also extends to the availability of drivers and their easy installation as well as regular software updates and their adaptation to new hardware configurations.

5. No Fragmentation

Because proprietary software suppliers own all the rights of disposal in the source code of the software they develop, they can also prevent the software from becoming fragmented and incompatible versions from emerging. There are also incentives for them to maintain compatibility with as many of the previously installed versions as possible and thus to protect and increase the network effects for users.

6. Variety of Application Software

There is an almost infinite number of applications available for proprietary software platforms. It is ensured that these applications always function equally well in different hardware configuration on the specified platforms.

The **strengths of open-source software** are based on the following points:

7. Customizability for Technically Inclined Users

Open-source software provides unlimited software customizability because its code is open source. However, the only users who benefit from this advantage are those who can draw on adequate know-how in order to customize it. Software customizability is hardly an issue for the mass market as the users do not have the necessary technical know-how.

8. Fast Spread of Knowledge

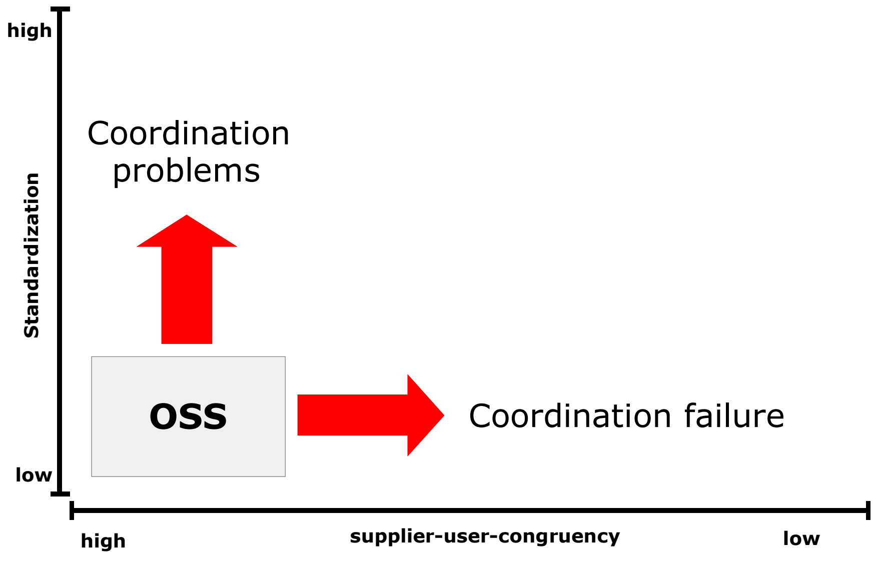
Because of the absence of ownership rights and the free accessibility of the source code, knowledge can spread fully and more easily in the open-source model. Knowledge produced in the open-source model is immediately available to all other developers. It must be noted, however, that the absence of ownership rights also leads to a lower production of knowledge.

9. Signaling

Open-source software discloses the development contribution made by each programmer. Because of that, open-source programming is better qualified to publicly signal a developer's skills. However, various proprietary software products also name the developers involved.

The differing strengths of the two software models are reflected in their respective product offerings. Because of these strengths, there are certain product groups in which each software model is at an advantage. Below is a closer examination of the importance of the standardization and developer orientation attributes for the software offering and a graphic depiction in Illustration 16.

Illustration 16: Application Fields of Different Software Types



The higher the level of **standardization** and the greater the compatibility requirements for software, the more likely will coordination problems occur in the open-source model. For a standard to become established and provide compatibility, it requires the kind of coordinated action that is more likely to be achieved within the structures and incentives of the proprietary model.

As regards **developer orientation**, developers and users can be one and the same people or they may belong to different groups. If users are also developers, user interests are automatically developer interests. If users are not developers, the developers' being guided by their interests will not necessarily lead to a product that also corresponds to the users' interests. As a result, there are mounting coordination problems in software engineering with respect to the coordination of user interests and product supply. In the proprietary model, the market mechanism provides for the coordination of interests. The wider the gap between users and developer groups, the more serious is the coordination failure in the field of open-source software.

Open-source software is therefore able to produce an adequate product supply for individual areas (little need for standardization, high supplier-user congruence). In the past, it has proven to be successful particularly in areas in which the attributes described above were of only minimal relevance. Open-source software has advantages with respect to products for which users also have developer know-how, for which standards and compatibility play a minor role and for which no fundamental innovations are required. The open-source model is not, however, qualified for the mass market. The proprietary model is clearly superior, especially with regard to long-term sustainability. It is better able than the open-source model to identify customer wants and to bring forth innovations.

Another limiting factor is the dependence of the open-source model on the proprietary software market. The discussion of developer motives has shown that one motive for participating consists of signaling incentives towards the proprietary software market. The reputation gained in an open-source project provides utility only if it can be exploited monetarily. Therefore, a commercial market has to exist in which the developers are remunerated for their efforts. Companies have to generate profits in this market that enable them to employ a large number of developers. If open-source developers do not have an opportunity to use their reputation in a proprietary market, they will lack a major incentive for contributing to open-source projects. As a result, the willingness to participate in open-source projects may decline as the proprietary market shrinks.

Even the existing commercial business models based on open-source software do not justify the long-term success of open-source software or its possible fitness for the mass market. As has been argued, the success of these business models hinges on complementary products being marketed, while this strategy results in economically inferior solutions overall in all the areas examined in this paper. The incentives for investing in the actual development of open source are also limited in commercial business models.

From a regulatory point of view, the open-source development model is therefore not a suitable substitute for the proprietary model. For the bulk of the software market, the proprietary model is better qualified to bring forth needed products. There still are numerous initiatives promoting the use of open-source software. The sections below examine whether such a promotion makes sound economic sense.

2.4.2 Motives for Promoting Open-Source Software

Lately, more and more governments have begun promoting the use of open-source software.[144] Because the government itself is a big demander of software, open-source software can be promoted by deliberately making procurement decisions in favor of open-source software. Even if a neutral cost comparison (TCO, total cost of ownership) comes to the conclusion that, including all relevant costs, the use of proprietary software is overall cheaper, a decision is nevertheless made in favor of open-source software.

For example, the decision of the city of Munich in favor of Linux is not based on economic reasons but rather on "qualitative-strategic" ones. The study conducted by the consulting firm Unilog arrives at the recommendation that "updating the Microsoft products currently in use to the current XP versions constitutes the technically easiest and economically soundest alternative for LHM".[145] The monetary advantages of the Windows XP alternative outweigh the next-expensive equipment alternative by approx. 2.46 million euros in principal value and by approx. 1.76 million euros in total value. The cost advantage of Windows XP as opposed to a pure Linux solution amounts to approx. 11.9 million euros in principal value and approx. 11.6 million euros in total value.[146] From a business point of view, the proprietary solution is clearly superior. From a "qualitative-strategic" point of view, however, the open-source solution is considered more advantageous. It is expected to ensure a higher level of basic security and to reduce manufacturer dependence, as the software products *"are not developed by one manufacturer but by an independent group comprising many developers"*.

Similarly, the German "Bundestux" initiative, which lobbied for the use of Linux in the run-up to a decision for new IT infrastructure for the German Parliament, did not view cost considerations as a key criterion in the procurement decision. The initiative called the "*introduction of a free operating system in German Parliament"*

[144] There are currently over 60 government initiatives, studies and statements in 25 countries aiming to step up the use of open source. Concerning this, see http://www.softwarechoice.org. HAHN (2002) also offers an overview of the different governmental activities.

[145] Cf. UNILOG INTEGRATA (2003), p. 29.

[146] Ibid, p. 19.

a *"necessary signal for Germany for regulatory, competition and location-policy reasons".*[147]

The feasibility study conducted by the consulting firm INFORA ultimately recommends using Linux in Parliament only in the field of e-mail servers and as a groupware solution. According to INFORA, one would be better served with Windows with respect to all other server services and desktop applications because open-source software still offers insufficient functionality for end users. Parliament ignored this recommendation and decided to deploy Linux on the file and print servers, too, and to have the directory service based on Linux, even though this solution costs some 80,000 euros more than the approach favored by Infora. The reasoning behind this decision is that in future procurements, there would no longer be a restriction to Microsoft-compatible products and that therefore this additional cost would pay off quickly – *"moving into a strategically advantageous position with just a little more money"*.[148]

The following sections examine the extent to which the promotion of open-source software can be the state's mandate. In this examination, a distinction is drawn between regulatory, competition and location-policy reasons.

2.4.3 Promotion of Open-Source Software

2.4.3.1 Not a Regulatory Mandate

Regulatory policy is understood to include all regulations and acts that allow the economy to be organized in accordance with market and competition principles. In this context, it is essential to defend competition and to give individual economic participants the freedom they require for their economic activities. This requires drawing a reasonable line between the activities of the state and those of the private economy.

In a market economy, the state's function is to establish a general framework. Furthermore, the state only intervenes in the market process with regulations or its own economic activity if the efficiency of the markets is not guaranteed. Competition policy is part of regulatory policy. The regulatory analysis in this section focuses on whether there are special functional deficits in the software market that might justify state intervention. The section thereafter discusses

[147] Cf. www.bundestux.de

competitive reasons for state intervention. It examines whether the positions in the software market necessitate intervention and whether the promotion of open source would be an appropriate way of intervening.

From an economic point of view, two criteria must be satisfied in order to justify state intervention in a certain market:

- A market failure exists in a market.
- The state can provide an efficient and inexpensive solution to this failure and the benefit generated by the elimination of this failure is greater than the accruing costs.

Ad 1: It must first be examined whether there is a market failure in the software market. Theoretically, a market failure could exist in the form of nonexcludability and network effects.

As the public goods theory has shown, there must be excludability for a market to even develop. **Nonexcludability** would be a consequence of insufficiently defined or not adequately enforceable ownership rights. If nonexcludability is present, the state can try to establish ownership rights or – if that is not possible – itself act as a producer or procurement agent. As the history of the software market has shown, the proprietary market evolved as a direct result of copyrights being established in order to exclude (or at least make legally excludable) users that were unwilling to pay. However, since the main criterion of open-source software is just this free availability, such an intervention is incompatible with the concept of open source.

There is a difference between "strong" network effects and "weak" network effects.[149] **Strong network effects** prevail over users' decisions. Even if a software product is superior, users will opt for the software that they expect will be the choice of all other users as well. Theoretically, a decision may come about that is in favor of an inferior technology, assuming that enough users expect a majority to opt for this new technology.

However, there are also mechanisms indicating that users will opt for the superior technology in spite of dominant network effects. It can therefore be assumed that users were primarily guided by the quality of the product at the time the network was formed. Subsequent users then jumped on the bandwagon. It is also possible

[148] Cf. No Author (2002). See: http://www.heise.de/newsticker/data/odi-27.02.02-000/

that users' decisions are indirectly synchronized through reading test reports and sharing experiences.

A market with strong network effects will produce a dominant network overall. If users have opted for the correct network, then this result would be economically efficient. If a better technology appears after a network has developed, it will make sense to switch to the better technology only if the costs associated with the move, i.e., the special investments in the switch and continuing education, are taken into account as well. Only if the utility derived from the new network is greater than the utility from the old technology, including the cost of switching, will it pay to switch networks.

If it is assumed that a new network would make economic sense because the current technology is inferior, the superiority of the new technology would have to be known before the technology could even establish itself in the market. Proving this fact is difficult, however, because neither state nor private institutions can decide beforehand which solution is the technologically superior one.[150] It is especially doubtful that government authorities would be able to make the correct technological decision for others.[151]

Even an objective assessment in favor of a new technology, regardless of how it came about, must factor in the investments made by every user in the form of training and the know-how need to use that technology. A switch makes economic sense only if the utility to be derived from the technology is so great that it exceeds the utility from the old technology plus the specific investments made.

If the state intervenes in a market with strong network effects, it may make a preliminary decision in favor of a technology and cause a major part of the market to lean towards that software. Therefore, an intervention in the market reduces the incentives for proprietary companies to invest in the development of software. On top of that, compatibility between proprietary and open-source software is restricted if open-source software is licensed under GPL. In that case, the open-source network would be created in parallel with a proprietary network. The promotion of open-source software in a market with strong network effects would

[149] Cf. SCHMIDT AND SCHNITZER (2002), pp. 15-19.

[150] Or, as stated more metaphorically in the original source, "the state and private institutions are both wanderers trying to find their way in the fog of future development". GRÖHN (1999), p. 54.

[151] "…governments do not have good track records at picking technology winners and losers" EVANS AND REDDY (2002), p. 71.

then establish a second network in a market in which a singular network would be economically superior.[152]

If there are **weak network effects**, competition may even decline because of the state's intervention. In this scenario, two parallel networks can develop, and there will be consumers that, despite the network effects, opt for one or the other platform. At the same time, there is a third group of consumers who do not have a preferred platform, and for whom there is therefore competition between the platforms.

If, in this environment, the state prefers open-source software, it influences just those users who do not have any basic preferences for or against a technology. This reduces the market size of the consumers for whom there is competition. As a consequence, the prices of proprietary software increase and investment incentives in the proprietary market are lessened.[153]

Ad 2: If the analysis uncovers any indications of an existing market failure, it does not automatically warrant an intervention by the state, as this does not prove that the state is actually in a position to bring about improved, let alone optimal allocation. There are several reasons why government action is not warranted. One can safely assume that the state's decision-makers do not have the right information (household preferences, cost structures of companies, development potential of new technologies) in order to improve allocation. On top of that, government countermeasures often change the incentive structure for the private sector. It is hard to determine whether these incentives can be influenced so as to improve allocation. Correcting the market failure often implies that market allocation decisions are supplanted by government decision-making processes. Experience has shown, however, that in many cases the state's decision-makers are guided not only by overall economic efficiency but – because of incentive structures within the bureaucracy – by their own interests as well.[154]

As such, there is no evidence that because of a market failure, an intervention in favor of open source would be possible. Instead, state intervention might lead to reduced competition, to diminishing network effects or to a leaning of the market towards a technology that is not necessarily superior.

[152] Cf. SCHMIDT AND SCHNITZER (2002), pp. 26-27.

[153] Cf. SCHMIDT AND SCHNITZER (2002), pp. 27-28.

[154] Cf. FRITSCH, WEIN AND EWERS (2003), pp. 83 et seq.

2.4.3.2 Not a Competition Policy Mandate

The promotion of open-source software is sometimes also considered a competition-policy measure. The discussion in this case revolves primarily around the market position of Microsoft. Promoting a second operating-system platform, it is argued, will curtail the market position of the Windows platform and reduce dependence on one supplier. The following discussion first examines briefly how to evaluate the competitive situation in the software market. This is followed by an examination of whether the promotion of open source constitutes an adequate competition-policy measure.

In evaluating competition in the software market, it is first necessary to define the market. In this case, promoting open source targets the market position of Windows in the desktop operating-systems market and the market position of the Office suite in the application-software market. The competitive situations in the desktop and application market are certainly very different. The difference is even more marked in the server market, where different products compete with each other. Promoting the use of open-source software would therefore not impact solely a certain position in a market segment but would have far-reaching consequences for other, highly competitive markets.[155]

As the discussion of the characteristics of software has shown, competition in the software market is subject to its own unique characteristics. On the one hand, they promote the formation of a certain market position; on the other, they limit the opportunities for fully utilizing any pricing leverage.[156] As previously stated, the software market is often characterized more by competition *for* the market than by competition *in* the market.

Nor is the size of the entry barriers to the software market entirely certain. Because development costs may represent sunk costs for an established supplier, this supplier might set his or her market price so as to prevent competitors from entering the market. In addition, the described network effects will tend to shore up the market position of an established supplier, particularly if switching costs are high. There are, however, several factors that limit switching costs. For example, in a market characterized by a steady inflow of new customers, market entry could also consist of competition for these new customers, who would not incur any costs

[155] Cf. on competitive aspects also EVANS (2002), pp. 44 et seq.

[156] Cf. on considerations regarding competition SCHMIDT UND SCHNITZER (2002), p. 7.

for switching. "Network bridges" may also be built in order to make different platforms compatible.[157] There are also incentives for an established supplier to continue developing his or her products, since, software being a product that does not suffer from wear-and-tear, once the market has been saturated, profits can be only generated if new products trigger repeat purchases.[158]

So far, these points indicate only that an assessment of the competitive situation must be very multifaceted und must not be based solely on the market share in a given segment. If a situation requiring intervention is identified, one must next determine whether the promotion of open-source software is an adequate tool for intervention.

If a supplier holds a powerful position in the market, there are any number of measures and institutions provided for in German and European antitrust laws that can intervene. If the competitive situation is examined and a violation of competition law is identified, there are any number of different measures of varying degrees of severity. However, none of the measures call for the state to specifically promote a competitor or an alternate technology. Nor do German and European anti-trust laws provide for pro-active interventions in favor of one side of the market; rather, these anti-trust measures are designed to stop anti-competitive behavior and to reestablish the requisite latitude for all market forces. It is then up to the market to make the final decision whether a given supplier can establish itself or not. Promoting a given supplier or technology in a competition-policy measure simply ignores the fact that the government does not even have the data needed to make an informed decision. Competition rules can only be protected if they abide by clear, transparent guidelines that focus on the rules and not the market result. Nonetheless, intervention in favor of given technologies – motivated mainly by the government's lack of information – is doomed to encourage private participants to use the state for their own particularist interests under the false guise of competition policy.

Nor do the most recent discussions concerning competition policies and their impact on Windows, focus in the main on how to establish a second platform. (Indeed, the network effects may make it economically efficient to have just one platform.) Rather, the competition issues deal with giving other providers access to

[157] For example, the offer of the Microsoft Office package makes a move between the Windows and Apple operating system easier for the Apple platform, as files have to be usable on both systems.

[158] Cf. GRÖHN (1999), p. 140.

the platform, expanding platform functionality, and integrating and tightly bundling individual applications to this platform. Even where these issues do indeed reveal anti-competitive behavior, they do not constitute adequate justification for replacing one platform with another.

It is often argued that promoting an alternate platform would reduce dependence on one supplier and combat the emergence of a "monoculture".[159] In principle, demanders will look favorably on any situation where they are not dependent on any one supplier. However, one should also consider that promoting different platform suppliers in the software market means that the positive network effects will be weakened. Thus one must weigh the advantages of being less dependent on one supplier against the disadvantages of using different platforms.

Nor is it clear whether opting for open-source software actually reduces dependencies. It might merely shift them. For example, if the demander has opted for a certain software product, he or she will again be dependent on the new software supplier. Because the same network effects apply to open-source software as to proprietary software, open-source software may also create monopoly situations with lock-in effects.

And although open-source users are not dependent on a given *producer*, they are still somewhat dependent on their particular *distribution.* The more specific the supplier's installation knowledge is, the greater the switching costs will be for the user. That means that all network effects and switching costs prevalent in proprietary software would also be present in open-source software. It thereby becomes impossible to reduce these fundamental dependencies. Furthermore, the examination of commercial open-source software models has shown that commercial suppliers use open source to finance the investments made in open source with profits from complementary products and services. If a commercial supplier succeeds in closely linking his or her open-source offering with this complementary product – and that is necessary if open source is to lead to commercial success –, then the dependencies will move away from the open-source product to the complementary product.

[159] Cf. ZYPRIES (2001).

2.4.3.3 Not a Location-Policy Mandate - Open Source and SMEs in the Software Market

The promotion of open-source software is occasionally also considered a competition-policy measure. The state's decision to demand a certain technology is calculated to generate a critical mass of demand so as to support the domestic suppliers of that technology. Such a decision is informed by the belief that, without this demand decision, a sector cannot develop at all or not as well and that the state can encourage the market development as an initial demander. In addition, the state expects more favorable effects for the national economy from promoting a technology developed, installed and serviced by local companies than from a technology that is developed entirely in a different country. This section therefore examines to which extent the state is a suitable agent for boosting the national economy by promoting open-source software.

Redirecting production flows into the domestic economy is an age-old commercial strategy. It basically contradicts the principle of global division of labor and ignores the current production structure in the software market. Just as division of labor and specialization within a national economy increase the general level of economic prosperity, so does international division of labor increase prosperity. Germany, as one of the world's biggest exporters, benefits considerably from this division of labor. Because the software market is horizontally structured, large portions of the value chain are covered by domestic companies even when software is imported. For example, 76,000 Microsoft-related jobs have been created at Microsoft's Certified Partners alone.[160]

As the analysis in section 2.3 has shown, the open-source model is, at its core, based on a deliberately nonmarket coordination mechanism. This mechanism is inferior in many respects, not only with respect to how well it performs market functions. If software engineering takes place outside the market, then no economically relevant transactions occur in this link in the value chain. If the software is (or has to be) available free of charge, its development – unlike in the proprietary software market – does not generate profit, income, jobs or taxes.

If the open-source community chooses not to price the software it produces, developers cannot be paid from the profits generated with the finished products. Complementary strategies (generating profits not with the software but with additional services or products) are not economically capable of balancing out the loss of value-added caused by the eradication of software-development profits (see also section 2.3, especially item 2.3.5.3). Though developers do make a contribution in the open-source model, this contribution is not rewarded by an economically relevant transaction. If proprietary software is replaced with open-source products, the jobs associated with this proprietary software are lost as well. Software developers, who as employees of domestic software companies are part of the economic value chain, are consequently squeezed out of the market.

In the open-source model, software development will entail economically relevant transactions only if developers are paid directly for programming as services rendered (engineering of custom software). This, however, is no different from the proprietary software market and can therefore not claim the advantages (open-source spirit) expected from the open-source model. As such, the development of open-source software does not, at its core, only take place outside the proprietary market of goods; it also takes place outside the labor market and consequently has no positive effects, and net negative effects.

According to the statistics on services, software development accounts for sales of 19.4 billion euros, 8,806 companies and 131,356 jobs in Germany. 99.6% of these companies are SMEs in the IT sector. The loss of a part of the value chain and of the associated jobs and opportunities to generate profit therefore hits SMEs in the IT sector particularly hard because, far from offering SMEs any new business opportunities, open-source software only offers some opportunities that are already present in the proprietary software market. The resultant effect on the national economy is not additive but substitutive.

If, when programming packaged software, a medium-sized software supplier is faced with the choice of developing for the open-source model or for the proprietary market, then the proprietary model offers an opportunity to recover the investments made in development by selling the designed programs. In the open-source model, however, the supplier can generate profit only by selling services in connection with the program. The more the open-source model spreads, the smaller the market becomes in which software can be exploited commercially. Thus in terms of location policies, supporting open source is not only an unsuitable means of promoting SMEs in the IT sector; it is inherently harmful, as open-source

[160] Cf. KOOTHS, LANGENFURTH AND KALWEY (2003). On the importance of the proprietary IT sector in various countries, see SMITH (2002), p. 79.

software robs the software market of the economic basis it needs to realize profits and create jobs. The German IT market is efficient and, with a share of two percent in the entire value chain, its importance to the national economy should not be underestimated. Open-source software gradually undermines this market.

Bibliography

BACKHAUS, K. (1999): Happy Engineering, in: managermagazin, issue 8, 1999, pp. 130-133.

BALZERT, H. (1996): Lehrbuch der Software-Technik: Software-Entwicklung, Heidelberg.

BERLECON RESEARCH (2002a): Free/Libre Open Source Software: Survey and Study, Use of Open Source-Software in Firms and Public Institutions Evidence from Germany, Sweden and UK, FLOSS Final Report - Part 1, Berlin, July 2002.

BERLECON RESEARCH (2002b): Free/Libre Open Source Software: Survey and Study, Firms' Open Source Activities: Motivations and Policy Implications, FLOSS Final Report - Part 2, Berlin, July 2002.

BERLECON RESEARCH (2002C): Free/Libre Open Source Software: Survey and Study, Basics of Open Source Software Markets and Business Models, FLOSS Final Report – Part 3, Berlin, July 2002.

BERLIOS (2003): http://openfacts.berlios.de/index.phtml?title=Open-Source-Lizenzen (October 27, 2003)

BITKOM (2003A): SoftwareTag 2003 Berlin-Brandenburg, Rahmenbedingungen für einen erfolgreichen IT-Mittelstand, Thomas Mosch.

BITKOM (2003B): ITK-Branche geht offensiv in die nächsten Jahre, BITKOM Press Release dated September 23, 2003.

BLANKART, C. B. UND KNIEPS, G. (1992): Netzökonomik, in: Jahrbuch für Neue Politische Ökonomie, vol. 11, pp. 72-87, Tübingen.

BORCHERT, M. UND GROSSEKETTLER, H. (1985): Preis- und Wettbewerbstheorie – Marktprozesse als analytisches Problem und ordnungspolitische Gestaltungsaufgabe, Stuttgart u.a.O.

BOSTON CONSULTING GROUP (2002): Hacker Survey, Release 0.3, Lakhani, K. R., Wolf, B. and Bates, J., LinuxWorld Presentation, January 31, 2002.

BURR, W. (1995): Netzwettbewerb in der Telekommunikation: Chancen und Risiken aus Sicht der ökonomischen Theorie, Wiesbaden.

COLECCHIA, A. AND P. SCHREYER (2001): ICT Investment and Economic Growth in the 1990s: Is the United States a Unique Case? A Comparative Study of Nine OECD Countries, STI Working Paper 2001/7, OECD, Paris.

EITO (2003): European Information Technology Observatory 2003, 11th ed.

EVANS, D. AND REDDY, B. (2002): Government Preferences for Promoting Open Source Software: A Solution in Search of a Problem, National Economic Research Associates, Cambridge, Mass., April 2002.

EVANS, R. (2002): Politics and Programming: Government Preferences for Promoting Opne Source Software, in: Government Policy towards Open Source Software, Editor: Hahn, R. W., pp. 34-49.

FEHR, E. AND SCHMIDT, K. M. (2001): Theories of fairness and reciprocity : evidence and economic applications, Münchener wirtschaftswissenschaftliche Beiträge.

FRANCK, E., JUNGWIRTH, C. (2002): Das Open-Source-Phänomen jenseits des Gift-Society-Mythos, in: Wirtschaftswissenschaftliches Studium : Wist, Vol. 31, 2002, Issue 3, pp. 124-129.

FRITSCH, M., WEIN, T., EWERS, H.-J. (2003): Marktversagen und Wirtschaftspolitik, 5th ed., revised and supplemented.

GERMAN FEDERAL STATISTICAL OFFICE (1999): Classification of Branches of Economic Activity with Explanations, 1999 Edition (WZ93) (German edition).

GERMAN FEDERAL STATISTICAL OFFICE (2001): Sales-Tax Statistics, Technical Series 14, Series 8 (German edition).

GERMAN FEDERAL STATISTICAL OFFICE (2002A): Services in Germany, Results of New Statistics – Year 2000, Wiesbaden (German edition).

GERMAN FEDERAL STATISTICAL OFFICE (2002B): National Accounts, Input/Output Tables, 1991 to 2000, published July 2002, Wiesbaden (German edition).

GRASSMUCK, V. (2002): Freie Software zwischen Privat- und Gemeineigentum; Bundeszentrale für Politische Bildung, Bonn.

GRÖHN, A. (1999): Netzwerkeffekte und Wettbewerbspolitik - eine ökonomische Analyse des Softwaremarktes, Tübingen.

GROSSEKETTLER, H. (1991): Die Versorgung mit Kollektivgütern als ordnungspolitisches Problem, in: Ordo, Jahrbuch für die Ordnung von Wirtschaft und Gesellschaft, Vol. 42, pp. 69-89, Stuttgart.

GROSSEKETTLER, H. (1995): Öffentliche Finanzen, in: D. Bender u.a. (Hrsg.), Vahlens Kompendium der Wirtschaftstheorie und Wirtschaftspolitik, Vol. 1, 6th ed., pp. 483-627, Munich.

HABER, G. AND GETZNER, M. (2003): Gesamtwirtschaftliche Effekte des Softwaresektors in Österreich 2003, Forschungsbericht, Version 2.31, September 5, 2003, Klagenfurt University.

HANG, J. AND HOHENSOHN, H. (2003): Eine Einführung zum Open Source Konzept aus Sicht der wirtschaftlichen und rechtlichen Aspekte, eine Studie im Rahmen des Projektes NOW: Nutzung des Open Source Konzeptes in Wirtschaft und Industrie, Siemens Business Services GmbH & Co, OHG, C-Lab, Paderborn.

HOCH, D. J. ET AL. (1999): Secrets of Software Success: Management Insights from 100 Software Firms around the World. Boston, Massachusetts, 1999.

HORST, E. (2003): Unser Bürgermeister soll schöner werden, Die Stadt München will zum Betriebssystem Linux wechseln, in: Frankfurter Allgemeine Zeitung, August 2, 2003, p. 36.

IANSITI, M. UND LERNER, J. (2002): Evidence Regarding Microsoft and Innovation, AEI-Brookings Joint Center for Regulatory Studies, Related Publication 02-4, Washington, D.C.

INSTITUTE FOR SMALL AND MEDIUM-SIZED ENTERPRISES (2002A): Mittelstand in der Gesamtwirtschaft – Anstelle einer Definition, Brigitte Günterberg und Hans-Jürgen Wolter, Institut für Mittelstandsforschung, Bonn.

INSTITUTE FOR SMALL AND MEDIUM-SIZED ENTERPRISES (2002B): Unternehmensgrößenstruktur in Deutschland nach Wirtschaftsbereichen und Rechtsformen, Brigitte Günterberg und Hans-Jürgen Wolter, Institut für Mittelstandsforschung, Bonn.

JANKO, W. H., BERNROIDER, E. W. N. UND EBNER, W. (2000): Softwarestudie 2000, Eine empirische Untersuchung der österreichischen Softwarebranche.

JOHNSON, J. P. (2001): Economics of Open Source Software.

KATZ, M. L. UND SHAPIRO, C. (1994): Systems competition and network effects, in: The journal of economic perspectives. Year 8, issue 2, pp. 93-115.

KLODT, H., LAASER, C.-F., LORZ, J. O., MAURER, R. (1995): Wettbewerb und Regulierung in der Telekommunikation, Kieler Studien 272, Tübingen.

KOOTHS, S., LANGENFURTH, M., KALWEY, N. (2003): Die Bedeutung der Microsoft Deutschland GmbH für den deutschen IT-Sektor (Economic Impact Study), MICE Economic Research Studies, Vol. 3.

KUHN, A. (2003): Die methodische Behandlung von Software in der Außenhandelsstatistik, in: Wirtschaft und Statistik, Issue 2/2003, pp. 121-125.

LEE, S., MOISA, N. AND WEISS, M. (2003): Open source as a signalling device : an economic analysis, Johann-Wolfgang-Goethe-Univ., Fachbereich Wirtschaftswiss., Working paper series.

LEHRER, M. (2000): From factor of production to autonomous industry: the transformation of Germany's software sector, in: Vierteljahrshefte zur Wirtschaftsforschung. Berlin : Duncker & Humboldt, vol. 69 (2000), 4, pp. 587-600.

LERNER, J. UND TIROLE, J. (2000): The simple economics of open source, NBER working paper series, no. 7600.

LERNER, J. und Tirole, J. (2002): The scope of open source licencing, NBER Working Paper Series, no. 9362.

LESSIG, L. (2002): Open Source Baselines: Compared to What? In: Government Policy towards Open Source Software, Editor: Hahn, R. W., pp. 50-68.

LÜNENDONK (2003): Lünendonk-Liste I 2003: Die Top 25 Standard-Software-Unternehmen in Deutschland, Lünendonk GmbH, Bad Wörishofen.

MENDYS-KAMPHORST, E. (2002): Open vs. closed : some consequences of the open source movement for software markets, CPB discussion paper.

MENDYS-KAMPHORST, E. (2002): Open vs. closed: some consequences of the open source movement for software markets, CPB discussion paper.

MICROSOFT (2003): Creating a Vibrant Information Technology Sector: Growth, Opportunity and Partnership, White Paper, updated May 2003.

MORNER, M. (2003): Open Source Software, in: Das Wirtschaftsstudium: wisu, Vol. 32 (2003), Issue. 3, pp. 318-321.

MUSTONEN, M. (2002): Why do firms support the development of subsitute copyleft programs? FPPE, Universität Helsinki.

NO AUTHOR (2002): See: http://www.heise.de/newsticker/data/odi-27.02.02-000/

NO AUTHOR (2003): Microsoft at the power point, in: The Economist, N. 8341 (Sep. 13th – 19th 2003), p. 61.

OECD (2002): Information Technology Outlook, ICTs and the Information Economy.

OSTERLOH, M. KUSTER, B. UND ROTA, S. (2002): Die kommerzielle Nutzung von Open-Source-Software: der Einfluss von sozialem Kapital, in: Zeitschrift Führung + Organisation, 2002, Issue 4, pp. 211-217.

RAHMEN-ZUREK, K. (2002): Internet economics and the organization of open source software-development In: Competition, environment and trade in the globalized economy / Dieckheuer, Gustav (2002), pp. 101-132

RAYMOND, E. S. (1998A): The Cathedral and the Bazaar. Elektronische Ausgabe unter: www.openresources.com/documents/cathedral-bazaar/.

RAYMOND, E. S. (1999): The Magic Cauldron http://www.tuxedo.org/~esr/writings/magic-cauldron/.

RAYMOND, E.S. (1998B): Homesteading the Noosphere, Elektronische Ausgabe unter www.firstmonday.dk/issues/issue3_10/raymond/.

RÖLLER, L.-H. (2002): Der Charme der freien Software, in: Frankfurter Allgemeine Zeitung, September 18, 2002, No. 217, p. 15.

SCHIFF, A. (2002): The economics of open source software: A survey of the early literature, The Review of Network Economics Vol. 1, Issue 1, pp. 66-74.

SCHMIDT, K. UND SCHNITZER, M. (2003): Public subsidies for open source? : Some economic policy issues of the software market, Centre for Economic Policy Research, Discussion Paper Series.

SHAPIRO AND VARIAN (1999): Information rules: a strategic guide to the network economy.

SMITH, A. (1776): Publication 2000: An inquiry into the nature and causes of the wealth of nations, Introduction by R. Reich, Annotations and Index by E. Cannan, New York, Modern Library, 2000.

SMITH, B. L. (2002): The Future of Software: Enabling the Marketplace to Decide, in: Government Policy towards Open Source Software, Editor: Hahn, R. W., pp. 69-86.

SPINDLER, G. (2003): Rechtsfragen der Open Source Software, im Auftrag des Verbandes der Softwareindustrie Deutschlands e. V.

STALLMAN, R. (2001): Free Software: Freedom and Cooperation, copy of the speech held at the University of New York on May 29, 2001
http://www.gnu.org/events/rms-nyu-2001-transcript.txt

UNILOG INTEGRATA (2003): Client Studie der Landeshauptstadt München, Kurzfassung des Abschlussberichts, abgestimmte Fassung, July 2, 2003.

VANBERG (1997): Die normativen Grundlagen von Ordnungspolitik. In: ORDO, Vol. 48, pp. 707-726.

WEBER, S. (2000): The Political Economy of Open Source, BRIE Working Paper 140, Economy Project Working Paper, Juni 15, 2000.

WEIZSÄCKER, C. C. VON AND KNIEPS, G. (1989): Telekommunikation, in: OBERENDER, P. (Hrsg.) Marktökonomie: Marktstruktur und Wettbewerb in ausgewählten Branchen der Bundesrepublik Deutschland, Munich.

ZYPRIES, B. (2001): „Linux-Tag: Abhängigkeit von Softwareherstellern verringern" Article in the newspaper "Die Welt" dated July 6, 2001,
http://www.bmi.bund.de/top/dokumente/Rede/ix_47733.htm
(Oktober 27, 2003).