Open Source Software and Economics[†]

Working paper (2002)

I. A. Kieseppä (SoberIT, Helsinki University of Technology)
E-mail: i.a.kieseppa@helsinki.fi
Home page: http://www.valt.helsinki.fi/staff/kieseppa

Abstract

This report summarizes a variety of economic aspects of open source (OS) software. The nature and the history of OS software are shortly considered in the beginning of the report. The economic questions which are raised by OS software are then divided into three groups, which are discussed separately. The questions of the first group are associated with the *microeconomic* analysis of the process of OS software production and, in particular, with the motives of the individuals who contribute to OS projects. Two fundamentally different ways of explaining these motives are distinguished. On the one hand, the motive for producing OS software can be simply the utility that the producer derives from using it herself. When this is the case, the economic models of the *private provision of a public good* are applicable to OS software production. On the other hand, the motives of the contributors can also be quite unrelated to such direct utilities; for example, the motive for contributing to an OS software project can be the wish to achieve a reputation as a successful programmer. The questions of the second group are associated with the *economic logic* of the process of producing OS software. We first consider the question how OS projects solve the problems which are due to the fact that such projects have a very large number of contributors who work in a parallel fashion, and then turn to a more systematic discussion of the typical features of successful OS projects. In this context we also consider some earlier analogies and some more general theoretical perspectives on OS software. The questions of the third group are *macroeconomic*, and they are concerned with the effects of OS software on the software market. Currently the literature on these topics is still rather rudimentary, and while considering them we shall rest content with discussing in general terms the software market, the ways in which commercial companies could react of OS software, and the effects that OS software has on software quality.

[†] While writing this report I have greatly benefited from the critical comments of Usva Kuusiholma and Jyrki Kontio.

Table of Contents

1. Introduction

Open code (OC) software is software which has been made available for free, typically on the Internet, and which may also be distributed freely. More precisely, what is available on the Internet is the code which is used by a computer which executes the software. Such software is often also called *freeware*. Freeware differs from *shareware*, which one also may download from the Internet for free, in so far that there are restrictions on how shareware may be used. Shareware is software which users are allowed to try before they decide whether they wish to buy it and, accordingly, while it is being downloaded its users are requested to accept an agreement according to which they use it only for a limited period of time if they do not pay for it.

The code which is available on the Internet in the case of freeware and shareware has normally not been written by a computer programmer. Rather, what a programmer produces is a *source code* which is translated into executable code by a computer. *Open source* (OS) software is software for which not only the executable code but also the source code has been made available to the public. Although the source code is not needed by a person who simply wants to make use of the software, it is quite essential for a user who wishes to make improvements to it. The *open source projects* which produce OS software have often a very large number of participants who make contributions to the software voluntarily, without monetary compensation.

As it is indicated in Figure 1.1, OS and OC software raise a number of interesting legal, economic, and technological questions. For an economist it is interesting to ask how the emergence of OS and OC software affects the market of ordinary proprietary software. In addition, the fact that programmers are willing to produce OS and OC software *for free* is both surprising and interesting for an economist. Since OS software products can be downloaded from the Internet for free, they are in the language of the economist *non-excludable public goods*, and such public goods are normally underprovided when contributing to them is voluntary. At the same time, the fact that the production and delivery of OS software are not associated with monetary transactions increases the difficulty of applying the tools of the economist to the OS software phenomenon.[1]

The most obvious of the juridical questions which the emergence of OS and OC software raises are concerned with the status of the open source *licenses*. Just like proprietary software, open source software is normally *licensed* to its users, and just like the licenses of many commercial software products which are available on the Internet, the OS licenses are most often "clickwrap" licenses. This means that a user who loads an OS product from the Internet is usually requested to express her acceptance of the terms of the license with a mouse click. One can ask whether this form of agreement is, in general, legitimate. It can also be asked whether, more specifically, the terms of OS licenses are legally enforceable. However, the broad extent to which OS licenses and "clickwrap" licenses of other kinds are currently used seems to allow one to answer both of these questions affirmatively. The legal questions which are concerned with the *liabilities* that might arise from using OS and OC software are more interesting. Open source and open code software might have very many co-authors, and it is not always clear who, if anyone, has the liability if their erroneous functioning e.g. causes a

[1] Cf. Ghosh (2002).

serious accident. Such questions of liability are particularly difficult if the software has not been licensed by any organization, as it is the case when the software has been produced by a well-established OS project, but is e.g. OC software which has been made available on the Internet by an unidentified person. Accordingly, it might be problematic for a commercial company to use software which comes from a source which is unknown to it. In addition, a commercial company might, of course, wish to avoid using such software also because it wishes to avoid being associated with some software producers, for example for image reasons.

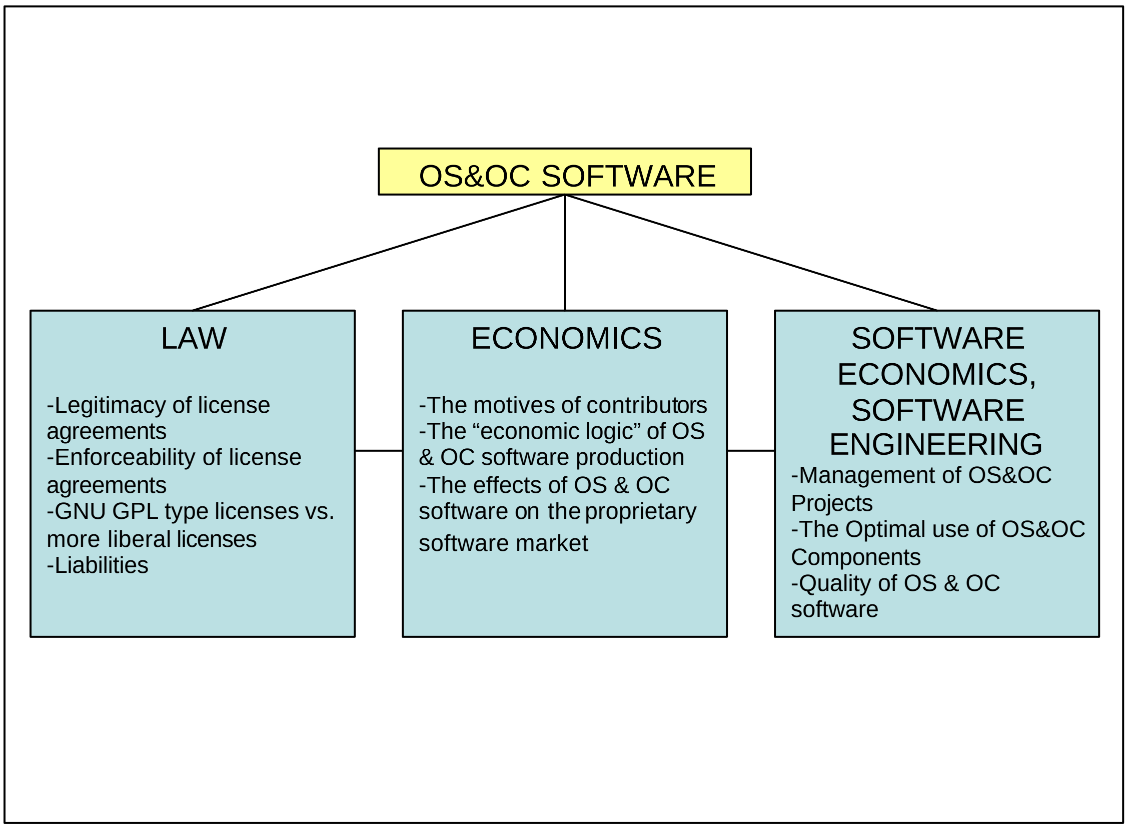
OS&OC SOFTWARE
LAW
-Legitimacy of license agreements
-Enforceability of license agreements
-GNU GPL type licenses vs. more liberal licenses
-Liabilities
ECONOMICS
-The motives of contributors
-The "economic logic" of OS & OC software production
-The effects of OS & OC software on the proprietary software market
SOFTWARE ECONOMICS, SOFTWARE ENGINEERING
-Management of OS&OC Projects
-The Optimal use of OS&OC Components
-Quality of OS & OC software

Figure 1.1. Aspects of OS and OC software.

Finally, many interesting problems of *software engineering* and *software economics* are associated with the *management of an open source project*, in which a large number of programmers volunteer in constructing software together. The managers of an OS project must organize the work of many individuals into a coherent whole and prevent the project from forking, i.e. prevent a development in which many incompatible versions of the OS product emerge. For example, the OS operating system *Linux* is an extremely complicated public good, which has resulted from the cooperation of a very large number of individuals who have contributed to it without any direct compensation for their efforts[2] and, accordingly, Linux has

[2] Cf. Weber (2000), p. 4.

been sometimes been called the " 'impossible' public good".[3] On the other hand, there also important software engineering issues which a programmer who *utilizes* OS software in her work has to consider. OS software products are often *software components* which are used as parts of larger software products, and a software producer who uses them has to weight the time that it takes her to search for them on the Internet and to verify that they function properly against the time that it would take her to produce similar components herself.

Steven Weber has divided the issues which an economic explanation of an OS software project must cover into three groups. Firstly, such an explanation must answer the question what the "core microfoundations" are, i.e. it must give an account of the motives of the individuals who contribute to the project. Secondly, the explanation must describe the "economic logic" of OS software production and, thirdly, the explanation must answer the question concerning "the social and political structures" which lead to successful OS software development in the context of given the motivations of the individuals and the "economic logic of software creation".[4]

In Section 2 below we shall shortly consider the history of OS software and have a somewhat closer look at two OS software projects, and we shall then move to a discussion of the economic questions which OS software raises. Similarly with Weber, we divide these questions into three groups. The questions of the first group are associated with the *microeconomic* analysis of the process of OS software production and, in particular, with the motives of the individuals who contribute to OS projects. The questions of the second group have to do with the *economic logic* of the process of producing OS software. Differing somewhat from Weber's classification, as our third group of questions we consider *macroeconomic* questions which are concerned with the effects of OS software on the software market. We shall discuss questions belonging to these three groups in Sections 3, 4 and 5 below.

[3] Kollock (1999), p. 230.

[4] Weber (2000), pp. 19-20. In ibid. Weber distinguishes these elements in the context of an explanation of the Linux project.

2. On the History of OS software

2.1 The Emergence of OS software

In Lerner – Tirole (2002) Josh Lerner and Jean Tirole divide the history of "cooperative software development" into three phases. The first of these is the period from the early 1960s to the early 1980s. Somewhat similarly with what currently happens in OS software projects, during this period software was transferred between institutions either freely or for a nominal charge, and the sites which installed it could develop it further by making innovations which were shared by others. At this time not only academic institutions but also corporate research facilities participated in such sharing.[5]

The second of the phases that Lerner and Tirole consider lasts from early 1980s to the early 1990s. This period is distinguished from the first one by the fact that during it one tried to formulate "ground rules" for cooperative software development.[6] An early formulation of such rules was given by *Richard Stallman*, a programmer working at the Artificial Intelligence Laboratory of the MIT. In the early 1980s the MIT licensed software to a commercial firm under such terms that its own employees, who had contributed to the development of the software, were prohibited from using the source code that they had themselves written.[7] Richard Stallman gave a rather extreme formulation for the dissatisfaction of these programmers by putting forward the idea that the practice of selling software or licensing it under an ordinary proprietary license is, in general, unethical. He has formulated this point of view e.g. as follows:[8]

> "The idea that the proprietary software social system – the system that says you are not allowed to share or change software – is antisocial, that it is unethical, that it is simply wrong, may come as a surprise to some readers. But what else could we say about a system based on dividing the public and keeping users helpless?"

Motivated by such ethical views, Stallman started in 1984 a project whose aim was to create an open source operating system called GNU which resembled the already available operating system UNIX.[9] This project is often quoted as the first important example of an open source project. While discussing the history of this project Stallman states that "we needed to use distribution terms that would prevent GNU software from being turned into proprietary software".[10] In particular, Stallman wanted to prevent a situation in which other programmers could produce *modified versions* of the software that his project had produced and turn them into proprietary software which could be sold or licensed under ordinary proprietary terms. The tool which the GNU project used for this purpose has become called "*copyleft*". By "copyleft" one means the practice of on the one hand allowing anyone to use and modify the

[5] Lerner – Tirole (2002), pp. 200-201.
[6] Ibid., p. 201.
[7] Cf. von Hippel – von Krogh (2002), pp. 4-5.
[8] Stallman (1999), p. 55 -57.
[9] Ibid., p. 57.
[10] Ibid., p. 59.

software but on the other hand prohibiting the users of the software from restricting the rights of others to use its modified versions or its combinations with other programs.[11]

In the case of GNU software this idea has been implemented by licensing it under what is called the GNU General Public License (GNU GPL).[12] The license is given by an organization created by the GNU project, *the Free Software Foundation*, which has the copyright to the software which the GNU project has produced. The terms of this license state that, although one is allowed to create modified versions of the software to which it applies and to distribute such versions, such versions must be licensed under the same license.[13] This means that the Free Software Foundation will have the copyright also to them.

These features of the GNU GPL have sometimes been criticized, because they seem to endanger the intellectual property rights of proprietary software producers: for example, if a GNU GPL program is introduced into an environment which contains proprietary software, the conditions of the GNU GPL license might force the producers of proprietary software to turn their work into OS software. It has sometimes been claimed that the introduction of GNU GPL software might in this way lead to a virus-like spread of the loss of property rights. However, the conditions of other OS licenses are less restrictive than those of the GNU GPL.[14]

The third phase in Lerner and Tirole's account of the history of OS software lasts from the early 1990s until today. This period is characterized by "a dramatic acceleration of open source activity" which has been caused by the diffusion of the Internet.[15] According to Lerner and Tirole the characteristic features of this period include also a shift from the GNU GPL license to more liberal licenses, which reflects the influence of "pragmatists"[16] (as opposed to persons like Richard Stallman, whose motives for producing OS software are more ideological). In addition, Lerner and Tirole also mention some negative developments as characteristic of this period. These include the 'forking' of some OS projects – i.e. developments in which there emerge several mutually incompatible versions of an OS program – and the lack of documentation, support, user interfaces, and backward compatibility in some OS projects.[17]

In this section we shall still have a somewhat closer look at two important OS projects, *Apache* and *Linux*. The other obvious examples of OS software which we could have chosen to consider include such popular software components as *Perl* and *Sendmail*, which are used while accessing the Internet.[18] We shall have a quick look at one more example of an important OS project in subsection 5.2 below, in which we discuss the decision of Netscape to turn its Internet browser, *Netscape Communicator*, into open source software. The example provided by Netscape is, obviously, particularly interesting because of the fact that the open source project which it has created is a project led by a commercial company, unlike e.g. Stallman's ideologically motivated GNU project.

[11] Ibid., p. 59-60.
[12] Ibid.
[13] Part 2b) of the "Terms and conditions for copying, distribution and modification" part of the GNU General Public License (http://www.fsf.org/copyleft/gpl.html, accessed on August 23, 2002). Cf. Lerner – Tirole (2002), pp. 201-202.
[14] Ibid., pp. 202-203. Cf. the article "Hackers rule" in the *Economist* (February 20, 1999; vol. 350, issue 8107), in which it is claimed that the license developed by Richard Stallman is "viral".
[15] Lerner – Tirole (2002), p. 202.
[16] Ibid., p. 204.
[17] Ibid., p. 203.
[18] Cf. ibid., pp. 210-212.

2.2 Two Examples: Apache and Linux

The development of the Internet server software "Apache" started in 1994. At that time most of the Internet servers made use of a server software whose source code was distributed freely by the National Center for Supercomputing Applications (NCSA). The NCSA had a development group which refined this software, making use of its consultations with its users. However, *Brian Behlendorf*, who was at that time a 21-year-old operator of an early commercial Internet server which utilized this software, and some of its other users grew increasingly dissatisfied with their cooperation with NCSA, and they decided to organize the collection and integrations of "patches" (i.e. pieces of source code which contained additions and modifications to the software) themselves. Collecting their patches together, they released a substantially different Internet server software package called Apache 0.8 in August 1995.[19] They introduced a very *modular* structure for their software, which enabled programmers to make modification to some parts of the program without affecting its other parts.[20]

At the time at which the Apache project was launched high-quality commercial software performing similar tasks with Apache was not yet available. Subsequently such commercial alternatives have been produced by both Microsoft and Netscape, but the OS software Apache has still a dominant position in the Internet server software market: according to Lerner and Tirole, Apache had in 1999 a market share of 55 % of all publicly observable web sites.[21]. At the same time Internet server software packages have grown into essentially more complicated products than they were when the Apache project was launched. It has from the beginning been their task to locate resources on the Internet, to send them requests from Internet users, and to send to the users the replies that the resources have given to their requests, but currently web server software packages perform also much more complicated tasks, like e.g. handling the authentication of users.[22]

Audris Mockus, Roy T. Fielding, and James Herbsled have discussed the Apache development process in Mockus – Fielding – Herbsled (2000) both in qualitative terms and by analyzing numerical data concerning it. They conclude from their analyses that the Apache development process "performs well on important measures".[23] The main points of their qualitative discussion of this process are reproduced below.

The Apache development process is guided by an informal organization called the "Apache group" (AG). The members of this group are volunteers who have made contributions to the project for an extended period of time. Only a part of the members of AG (usually 4 to 6 persons) are actively working on the Apache project at each point of time.[24] The development work of such "core developers" follows a well-defined pattern which begins with the discovery of a problem. The problem might have been reported in the mailing list of the developers, it might have become known via the *problem reporting system* of the Apache project into which

[19] Ibid., p. 207.
[20] Ibid., p. 207-208.
[21] Ibid., p. 209.
[22] Lakhani – von Hippel (2002), p. 4.
[23] Mockus – Fielding – Herbsleb (2000), p.263.
[24] Ibid., pp. 265.

anyone is allowed to send problem reports, or it might have been discussed in the Apache-related newsgroups of the USENET. However, the high "perceived noise level" – i.e. the experienced low quality of the discussion – in these newsgroups lowers the amount of attention that the members of the AG pay to the news that are published in them.[25]

When a problem has been discovered, a volunteer is searched for dealing with it. For obvious reasons the developers of Apache usually prefer to work on those parts of the code on which they have already previously worked and with which they are accordingly most familiar. When a developer has volunteered to work on a problem, he identifies some possible solutions to it. There is usually more than one solution to each given problem and when this is the case, the volunteering developer, as a rule, asks for the feedback of others on which of them should be implemented.[26] When an agreement concerning the appropriate solution has been reached, the developer implements the solution and tests it himself. After such tests a developer can proceed in either of two ways. He can either send the modifications that he suggests to what is called the Concurrent Version Control archive (CVS), or he can send the "patch" which contains the modifications to the developer mailing list for review, after which they can be committed to the CVS. The inclusion of such modifications in the official releases of Apache is managed by a *release manager*, who has also the task of evaluating, in more general terms, whether the software with its new patches is sufficiently stable to permit a new release.[27]

Although the Apache Group guides the development of the "approved versions" of Apache, it does not provide Apache *support*, i.e. assistance for those users of Apache who run into difficulties while using it. It is an interesting feature of the Apache project that an on-line Apache field support system has emerged despite of the lack of any "official support" from the Apache Group.[28] The persons who contribute to this support system by answering the questions from other users do it without any direct compensation for their work and, just like one can ask what motivates programmers to make contributions to OS software, one can wonder what motivates the persons who answer the questions of Apache users. *Karim Lakhani* and *Eric von Hippel* have conducted an empirical study which is concerned with this question, and they present in Lakhani – von Hippel (2002) a list of twelve reasons which have motivated the providing of information to other Apache users. Many of the items on this list resemble the reasons which we shall consider in Section 3 below, and which are often quoted as motivating OS programmers: the reasons that they mention include a belief in reciprocal help, the wish to enhance one's career prospects, the wish to promote OS software, and the feeling that answering the questions of other users is fun.[29]

A feature which our other example, the *operating system Linux*, shares with Apache is that its construction was initiated by a very young developer. When *Linus Torvalds* initiated the Linux project, he was a 21-year-old graduate student who wanted to build an operating system which could be distributed freely and which would resemble the operating system Minix.[30] The operating system Minix was a simplified version of Unix, and it was at that time widely used in computer science courses.[31] More specifically, Linus Torvalds announced on the Internet in October 1991 that he had written a free version of an operating system which resembled Minix

[25] Ibid., pp. 265-266.
[26] Ibid., p. 266.
[27] Ibid., pp. 266-267. (Cf ibid., p. 265.)
[28] Lakhani – von Hippel (2002), pp. 5-8.
[29] Ibid., p. 47.
[30] Lerner – Tirole (2002), p. 208.
[31] Moon – Sproull (2000), section "Linus Torvalds".

and that he would be willing to add functions written by others to it if also these were made freely available. From the perspective of our subsequent discussion of the significance of the OS community (see subsection 3.2 below), it in interesting to observe that in his original message Torvalds emphasizes his identification with such a community by stating that his program is “a program for hackers by a hacker”.[32]

Originally, the licensing agreement under which Linux was released was rather restrictive: it required that all programs which were distributed or used together with Linux would be freely available. However, these restrictions were subsequently relaxed.[33] Currently Linux is licensed under the rather restrictive license GPL, but it is not requested by the license agreement that programs which are run using the operating system Linux would have to be covered by the same license. In other words, it is legitimate to produce also proprietary software which has been intended to be used together with the operating system Linux.[34]

The number of the users of Linux has grown rapidly. According to Lerner – Tirole (2002), Linux had approximately 100 users one year after the project had started, but already in 1994 there were half a million of them. Subsequently Linux has developed into a significant competitor of the Microsoft Windows operating system, and the estimates of its current number of users range from 7 to 16 million.[35] The number of the programmers who have contributed to Linux is difficult to estimate. According to Jae Moon and Lee Sproull, the published estimates of their number range from several hundreds to more than 40,000.[36] The large differences in these estimates are, of course, caused by the fact that the value of an appropriate estimate depend crucially on how substantial a contribution has to be in order to qualify as a “contribution to Linux”.

Linus Torvalds has himself discussed the development of Linux in some detail Torvalds (1999). In his own account of its development he emphasizes a number of successful engineering decisions that he made. In particular, he emphasizes the fact that he rejected the idea that a *portable* operating system – i.e. an operating system which can be used with many different computer architectures – should have what is called “microkernel-style architecture”.[37] Torvalds also views the highly *modular* structure of the operating system – i.e. the fact that it consists of separate components with well-defined tasks of their own – as essential for its success.[38] The explanations that others have given for the success of Linux have sometimes referred to exceptional programming skills of Torvalds but, as Moon and Sproull point out, his qualities as a manager of the project seem to have been at least equally important.[39]

As the Linux project gained momentum, Torvalds quickly moved from writing code to leading the project by assessing the contributions to it and arbitraging disputes between contributors. Subsequently a considerable part of such decision-making has been delegated to

[32] Torvalds's e-mail message from October 5, 1991 in which he announced that the free version of Minix had become available has been reproduced in Moon – Sproull (2000), section “Linus Torvalds”.
[33] Lerner – Tirole (2002), p. 208. Cf. Torvalds (1999), pp. 108-109.
[34] Torvalds (1999), pp. 108-109.
[35] Lerner – Tirole (2002), pp. 208-210.
[36] Moon – Sproull (2000), section “Introduction”.
[37] Torvalds (1999), pp. 103-104.
[38] Ibid., pp. 108. Cf. Moon – Sproull (2000), section “Linus Torvalds”.
[39] Moon – Sproull (2000), section “Linus Torvalds”. Moon and Sproull characterize the qualities of Torvalds as a leader by stating that he on the one hand keeps the authority to make decisions but, on the other hand, avoids giving orders and makes suggestions in a mild-mannered style of communication (ibid.).

other contributors, who are called “lieutenants” of the Linux project.[40] According to Moon and Sproull the two most important roles within the community of Linux developers are those of a *credited developer* and a *maintainer*. Since the 1994 release there has been “credits file” attached to the release which lists persons who have substantial contributions to the release. The maintainers role was formally acknowledged for the first time in February, 1996, in which a “maintainers file” was announced. The maintainers file contains names of designated maintainers who take responsibility for particular modules of the kernel.[41]

Another essential feature of the Linux development model was the establishment of a *parallel release structure* which was such that there were two kinds of releases of Linux, which were oriented at different audiences. The odd-numbered releases incorporated new features, and they were meant for developers who wanted to get feedback for their ideas. Accordingly, the users of such versions would find in them “bugs” (i.e. the programming mistakes) which they could report and fix. When an odd-numbered version had been tested and corrected in this way for a sufficiently long time, it was released as an even-numbered version, which had fewer bugs and which was oriented at the ordinary users of Linux, who wanted their operating system to be stable and reliable.[42]

Having shortly considered two successful OS projects, we now turn to the task of applying the tools of the economist to such projects. Presumably, the most obvious economic question which projects like Apache and Linux raise is the one which is concerned with the *motives* of the persons who contribute to such projects despite of the fact that they could also “free-ride”, i.e. make use of the available software without developing it further. This question is central for the *microeconomic* analyses of the phenomenon of OS software, which we shall consider in the next section.

[40] Lerner – Tirole (2002), p. 210.
[41] Moon – Sproull (2000), section “Community”.
[42] Ibid., section “Linus Torvalds”.

3. Microeconomic aspects of Open Source Software Production

In the language of the economist, OS software is a *public good*. By definition, this means that OS software is a *non-rival* good: the consumption of OS software by one consumer does not diminish the amount of OS software which is available to others. In addition, OS software is a *non-excludable* good in the sense that its producer is not able to determine who is able to make use of it.[1] Analogously with *street lights*, which are often quoted as an example of a non-excludable public good, and which are available to either everyone or to no one, also a piece of OS software which has been placed on the Internet can be used by anyone who wishes to download it onto her computer.

According to microeconomic theory, non-excludable public goods are associated with *market failures*, i.e. situations in which market mechanisms fail to allocate resources efficiently.[2] If street lights were financed by asking the persons who benefit from them to pay for them just like they pay for private goods, the amount of money that each individual would be willing to pay would, according to microeconomic theory, be smaller than the monetary counterpart of the benefit that they actually derive from street lights. This would be the case, because if each person made the decision whether to pay for street lights *separately*, every one could observe that almost exactly the same amount of street lights would be available to her independently of her contribution.

The fact that no such market failures seem to exist in the context of OS and OC software, since programmers are willing to produce it for free, seems to contradict standard microeconomic theory. Accordingly, the economists who have considered the phenomenon of OS and OC software from a microeconomic perspective have largely focused their attention to explaining the motives of OS&OC software developers. In this section we shall present a review of such explanations that have been suggested in the earlier literature.

The accounts that economists have given of the motives which make programmers produce additions and modifications to OS&OC software can be roughly divided into two groups. An explanation can be based on the utility which the developer of the software derives from using it herself, or on considerations of some other kind, like e.g. the effects that contributing to OS projects might have on the career prospects of the contributor. We shall consider these explanations in the first two subsections of this section. A closely related question is what motivates programmers to make the software that they have written available on the Internet. When the motive of producing OS software is claimed to be personal use, this question is in need of a separate answer. If a programmer has written an addition to OS software simply because she needs it herself, it is not clear for what reason she should make that addition available to all other users of the same software, who might include e.g. representatives of companies which compete with her company. A possible answer to the latter question is based on the *network externalities* which are associated with OS software. We shall consider such network externalities in subsection 3.3.

[1] See e.g. Gravelle – Rees (1994), pp. 525-528.
[2] Ibid.

3.1 OS Software Production as User-Driven Innovation

Although the phenomenon of OS is surprising for an economist, it is not quite unprecedented. As Lerner and Tirole point out,[3] "user-driven innovation" in which new technology is developed by sophisticated users has existed in other fields before the emergence of OS software. For example, in von Hippel (1988) Eric von Hippel mentions three innovation categories in which it is the user rather than the manufacturer who "recognizes the need, solves the problem through an invention, builds a prototype and proves the prototype's value in use".[4] These first of these innovation categories are the innovations in the fields of *scientific instruments*, and the other two belong to the field of electronics. More recently, von Hippel has pointed out that, just like the contributors to OS projects, also the practitioners of various sports like e.g. windsurfing form *innovation sports communities* in which some practitioners of the sports invent new sports equipment, and other practitioners "test" it, just like the users of OS software contribute to OS projects by testing the software and reporting bugs.[5]

According to von Hippel, two obvious reasons what motivates user-innovators are the fact that a manufacturer cannot know what users want as well as they themselves do and the fact that, even if they did, they would lack the incentive to match their wishes in every detail. The most obvious reason for the latter problem is that since manufacturers want to spread their development costs between many users, the products that they develop contain compromises between the wishes of different users. The former problem is partially caused by the fact that the information concerning the wishes of the users is "sticky" in the sense that it is costly to transfer it from the users to the developers.[6] There are, in general, at least two obvious reasons which might make technological knowledge "sticky" in this sense. On the one hand, technological knowledge is often "tacit", i.e. it is often partially practical know-how which has not been given an explicit linguistic expression, and on the other hand, it is often concerned with the specific features of one particular technological application.[7] The latter reason for the "stickyness" of technological information seems more important in the context of OS software. For example, it would be costly to bring the programmers who are employed by a company which produces proprietary software to know in a very detailed manner the circumstances under which each of the observed bugs in the software has been detected. A related problem is that the wishes of the users of software products change over time, and they are affected by the experiences that they receive while using the software.

Already before the emergence of OS software economists have produced models for the private provision of a public good, in which its production is motivated by utility that the user receives from using it herself. For example, such models are presented in Palfrey – Rosenthal (1984), in Bliss – Nalebuff (1984), and in Bergstrom – Blume – Varian (1986). Each of these three models is concerned with a situation in which each of a finite number of agents has to make a choice concerning the extent to which she contributes to the production of a public good. What is being analyzed in these three models is a *Nash equilibrium*, i.e. a situation in which the choice made by each agent is from her perspective optimal as long as none of the other agents make changes to the choices that they have made. In e.g. the model of Palfrey and

[3] Lerner – Tirole (2000), p. 3
[4] Von Hippel (1988), p. 25.
[5] Von Hippel (2001), pp. 82-83.
[6] Von Hippel (1994), pp. 430-432.
[7] Ibid. Cf. Rosenberg (1976), pp. 77-79.

Rosenthal there is a large number of different Nash equilibria. In this model it turns out that if all agents have decided not to contribute to the public good, each agent can separately reason that it is optimal for her to stick to this decision, since her contribution will not alone be sufficient for producing the public good. However, if a suitable number of agents has decided to contribute to the public good in Palfrey and Rosenthal's model, each of them can reason that it is optimal to stick to this decision as well, because in this case the public good will get produced if they contribute to it.[8]

None of the above models is particularly well suited for analyzing OS software production.[9] However, if one assumes that the production of OSS is motivated by the utility that its producer receives from using it herself, one can easily construct an economic model of OS software production by modifying such earlier models. A modified model of this kind has recently been presented by Justin Pappas Johnson.[10]

In Johnson's model there are a finite number of agents who are using OS software and who have to make a decision whether to develop some particular new application for the software. In the model the piece of software gets produced if at least one of the agents chooses to the develop it. The utility that the agents would receive from the application and the effort that they would have to put into producing it is assumed to differ from agent to agent in a random manner. There are several interesting questions concerning the OS software production process which can be addressed in the context of this model. The most obvious of these are associated with the effects of the number of the agents on the probability that the application gets produced. Since the number of potential contributors to OS projects is large – in principle, the potential "labor force" of an OS projects contains the "entire internet community of programmers"[11] - it is natural to ask what happens when the number of agents receives a very large value in Johnson's model.

An increase in this number, which Johnson calls n, has two kinds of effects on the probability that someone produces the new piece of software. On the one hand, if the probabilities with which contributors produce the software are thought of as fixed numbers, an increase in the number of contributors has, of course, the effect that the probability with which the software gets produced grows. However, the growth of n makes also the motivation that each programmer has for producing the software grow smaller, since each contributor can feel more confident that even if she does not produce the requested piece of software *some one else will produce it on her behalf* and that she will be able to benefit from it as a free-rider.

It is interesting to ask which of these effects will be stronger; i.e., it is interesting to ask whether the growth of the number of persons who are involved in the OS community will in Johnson's model tend to increase or decrease the probability with which software gets

[8] Palfrey – Rosenthal (1984), p. 174.

[9] In the model in Palfrey – Rosenthal (1984) the public good comes to existence only if a sufficient number of agents make an identical contribution to it. This is quite unlike the production of a modification or an addition to OS software, which each agent can by herself choose to produce. In the model in Bliss – Nalebuff (1984) the decisions concerning the *time* at which the contributions to the public good are made have a very essential role which they do not seem to have in the context of OS software production. Finally, in the model in Bergstom – Blume – Varian (1986) the magnitude of the contribution which the agents make to the public good is a continuous quantity. This assumption is reasonable when the contributions are sums of money, but quite unrealistic when the contributions are pieces of source code, because the length of a useful piece of source code cannot be arbitrarily chosen by its producer.

[10] Johnson (1999).

[11] Ibid., p. 13.

produced. Johnson's model yields neither the answer that this probability always increases nor the answer that it always decreases when an additional member is added to the OS software community.[12] It is, however, possible to show that, as *n* grows, the probability with which the software gets produced always approaches a fixed limiting value which greater than 0 but smaller than 1.[13] In addition, also the expected number of agents who choose to produce the software converges to a finite positive limit as *n* grows.[14]

The last of the above results has an interesting interpretation. It is clear that OS software production contains a wasteful element: when lots of programmers write OS software in an uncoordinated manner, it will sometimes happen that many programmers produce pieces of source code which have a similar function. The last of the above results means that in the context of Johnson's model there is a limit to such wastefulness. It means that the expected number of programmers who write similar pieces of code, of which only one would be needed, does not grow indefinitely as the population of programmers grows, but has some finite limit. It also turns out that in the context of Johnson's model *the expected welfare* of each agent increases as the number of the participants in the OSS project increases.[15]

The above discussion was concerned with a single contribution to an OSS project, but Johnson extends his model also to the analysis of a whole OS project which consists of a large number of small tasks. In the *non-modular* version of his model the OS software product gets developed only if one of the contributors has completed *all* of the tasks that it consists of. This version is a rather trivial modification of Johnson's earlier analysis, since in this case each contributor will choose either to perform all the tasks or to perform none of them. In the *modular* version of the model it is not necessary that the same person completes all the tasks of the project. In one of his theorems Johnson shows that it depends on the number of contributors whether more software gets written in the modular or in the non-modular case. More specifically, he shows that when the number of contributors and the number of required tasks are sufficiently large, the expected value of the number of completed tasks is larger in the modular case than in the non-modular case. However, the opposite will be the case if the number of contributors is small and the number of tasks is large.[16] The reason for the fact that the non-modular environment outperforms the modular one when the number of contributor is small is, of course, that in the non-modular case each contributor knows that her contribution will be worthless if she does not perform all the tasks of the project and that, since there are few other contributors, each contributor has little confidence in other contributors completing these tasks.

Also this result has an empirical interpretation which seems to correspond to the facts. It means that OS software development model, in which software is built from small "modules" which have been produced by different authors, functions efficiently only when there is a sufficient "developer base", i.e. when the number of programmers who contribute to the project is sufficiently large. Another interesting fact which, Johnson claims, his model

[12] Ibid., p. 12.
[13] Ibid., p. 14.
[14] Ibid., p. 17.
[15] Ibid., p. 13. The definition of welfare which Johnson implicitly uses in this result is such that the the welfare of an agent who chooses to produce the software is given by the difference between the agent's valuation of the new software and the costs that producing it has for him or her. On the other hand, if an agent does not produce the software but someone else does, his or her welfare is expressed simply by her valuation of the software, and if the software does not get produced at all, the value of each agent's welfare is 0. (Cf. ibid.)
[16] Ibid., p. 25.

explains is that some rather simple open source products – like e.g. word processors – do not get produced, while other much more complicated ones – like e.g. complicated network utilities – do.

Johnson's explanation for this fact is the correlation between the value that a software product would have for a potential contributor and the costs that its production would incur on her. The probability that the product does not get developed is large when this correlation is large and positive: in this case the users for whom the product would be most useful are at the same time the ones for whom its development would be the most costly task. On the other hand, when this correlation is negative, it is quite likely that the product does get developed: in this case the persons who are in the greatest need of the product are the ones for whom its production is most easy. A possible explanation for the fact that OS model has been more successful in producing network utilities than word processors is that OS word processors are products which belong to the former group, and that OS network utilities are products of the latter group.[17] In other words, OS word processors are products which would be most useful for those members of the OS community who are the least able to produce them, but OS network utilities are most useful to the ablest programmers.

The motives of OS programmers are for obvious reasons important also for the evaluation of the *quality* of OS products. *Jennifer Kuan* has suggested that if OS software is written for personal use, it can be expected to be of a higher quality than comparable proprietary software, and presented empirical results in which she finds support for the claim that this is, indeed, the case.[18] On the other hand, as e.g. *Gilles Saint-Paul* has pointed out, non-proprietary goods have been intended to fit the desires of their inventor rather than their other consumers, and they might for this reason be inadequate for other consumers.[19] It has also been argued that a commercial software company has a clear incentive to perform tests on its products, which has the effect of making their products more secure. On the other hand, in a "community-driven" software project testing procedures will be more *ad hoc*, and that a greater part of the burden of testing will fall to the users of the software.[20]

3.2 Some other Motives of OS Software Production

It is clear that any microeconomic analysis of the phenomenon of OS software will share some basic features with the model of Johnson which we considered above. Just like in the analysis of the production of any other good, in an analysis of OSS software the producers are viewed as confronted with the *costs* and the *benefits* that the production causes, and a rational agent is assumed to maximize the *utility* which she receives from production, given these costs and benefits. In the case of OS software the costs of production seem to be solely costs of the time that the programmer puts into working within the OS project. When an OS software producer works for free, her working time is not explicitly associated with a monetary value, but an economist can measure its value with its *opportunity costs*, i.e. the value of the other uses that her time could have been given.

[17] Ibid., pp. 21-22.
[18] Kuan (2002), p. 3.
[19] Saint-Paul (2002), p. 3.
[20] These views have been put forward by e.g. Craig Mundie, a representative of the Microsoft Corporation, in Mundie (2002).

The opportunity costs depend, among other things, on the demand for computer programmers on the labor market. Accordingly, as it has been pointed out by David Lancashire, a microeconomic analysis of OSS production makes it easy to understand why during the last ten years the "nexus of open source development appears to have shifted to Europe"[21] from the United States. As Lancashire states, in recent years per capita investments in information technology have in the United States exceeded those in Europe, and this has made the demand for highly-skilled computer professional rise in the United States. This implies that also the opportunity costs of OS software production have risen in the United States, making it less attractive to American programmers than it is to European programmers.[22] Lancashire's comparisons of Europe and the United States motivate an interesting question. It seems reasonable to assume that for many programmers in the developing countries the opportunity costs of OS software production would be still lower than they are for the European programmers who have started to participate in OS projects, and this makes it natural to ask why programmers from the developing countries do not produce more OS software than they do. However, in this report we shall not discuss this topic in a detailed manner.

Any microeconomic analysis of OSS production will have to take the opportunity costs into account in essentially the same manner, in the way suggested by Lancashire. However, the benefits of participating in OSS production which are mentioned in such analyses can be quite different. In Johnson's model the only motive for producing OS software was personal use, but there might be also other, quite different reasons for producing it. There has been a considerable amount of self-reflection within the community of OS software developers concerning such reasons. Above in Section 2 we already mentioned the ideological background of OS software when we considered the rather extreme views of Richard Stallman. Steven Levy has discussed this background in a more systematic manner in his book "Hackers", in which he discusses the development and the nature of the hacker subculture. He claims that the birth of the "way of life" of hackers was accompanied by the development of a specific "hacker ethic". He characterizes this new ethic by listing the following views and imperatives:[23]

> Access to Computers […] should be unlimited and total. […]
> All information should be free.
> Mistrust Authority – Promote Decentralization.
> Hackers should judged by their hacking, not bogus criteria such as degrees, age, race, or position.
> You can create art and beauty on a computer.
> Computers can change your life for the better.

It is clear that these views can serve as a motivation for contributing to OS software projects, which are decentralized and which enlarge the number of the people to whom computers are accessible. On a more general level we can, following Steven Weber, divide the motives for producing OS software which are mentioned in self-reflective accounts of the OSS community

[21] Lancashire (2001), Abstract.
[22] Ibid, section "Economic Theory Revisited". In Lancashire (2001) uses this observation as an argument against those analyses of the OS phenomenon in which it is discussed in purely in purely cultural rather than in economic terms.
[23] Levy (1984), pp. 39-45.

into four groups.[24] Firstly, there is a motivation in the normative views according to which proprietary software is unethical. Such views are represented most notoriously by Stallman.[25] Secondly, there is an "artistic" motivation: it has been claimed that hackers work like artists and that they are motivated by the "fun, challenge, and beauty" that they see in their work. However, as Weber readily points out, this claim does not explain the fact that, unlike most artists, OS producers give their art away for free.[26] As a third motive he mentions the commitment to oppose Microsoft, and as a fourth motive "ego-boo", which is short for "ego-boosting".[27]

This motive has been acknowledged by e.g. *Eric S. Raymond*, who represents a wing of the "hacker culture" which is quite different from that of Stallman.[28] According to him, "The utility function Linux hackers are maximizing is not classically economic, but is the intangible of their own ego satisfaction and reputation among other hackers".[29] He states that "voluntary cultures that work this way are not actually uncommon", and mentions "science fiction fandom", i.e. the subculture of the science fiction fans, as another example of such a voluntary culture. According to him this culture has "long explicitly recognized 'egoboo' (ego-boosting, or the enhancement of one's reputation among other fans) as the basic drive behind volunteer activity".[30] However, Steven Weber points out, it seems that participation in OS projects might lead not only to "egoboo" but also to ego damage – this might happen, for example, when the patch which a programmer has written gets rejected – and it is difficult to understand on the basis of Raymond's explanation what makes programmers contribute to OS projects when this has happened.[31]

Josh Lerner and Jean Tirole have discussed in a recent paper, Lerner – Tirole (2002), the motives of OSS producers in more conventional economic terms. They divide the *net benefit* that a contributor to an OS project receives from her contribution into the *immediate payoff* and the *delayed payoff*. The immediate payoff is the difference between the current benefit and the current cost and, as it was explained above, the current cost of the contribution is measured by the opportunity cost of the time of the programmer. The immediate benefits might include a monetary compensation, if the programmer works for a commercial firm which is involved in producing OS software, direct benefits from personal use, and benefits that she receives from others using the program.[32]

According to Lerner and Tirole, the delayed payoff contains two "distinct, although hard-to-distinguish", incentives. These are the *career concern incentive* and the *ego gratification incentive*. These are both associated with the wish to receive recognition for one's work, but the audience from which one wishes to receive it might be different in the two cases. Lerner and Tirole call the totality of the incentives to receive recognition the *signaling incentive*. They state that the signaling incentive "is stronger a) the more visible the performance to the

[24] Weber (2000), p. 20.
[25] Stallman (1999), p. 54. (Cf. p. ? above.)
[26] Weber (2000), p. 20.
[27] Ibid.
[28] Cf. Raymond (2001), p. xiii.
[29] Ibid, p.53.
[30] Ibid.
[31] Weber (2000), p. 21.
[32] Lerner – Tirole (2002), p. 14.

relevant audience (peers, labor market, venture capital community), b) the higher the impact of effort on performance, and c) the more informative the performance about talent".[33]

Lerner and Tirole contrast the incentives which, according to them, motivate programmers to write OS software with the motives for writing proprietary software. Since also the writing of proprietary software might be associated with the benefits that were mentioned above and since programmers are offered a salary to write it, the production of OS software can be explained in Lerner and Tirole's framework only by claiming that these incentives are stronger in the case of OSS than in the case of proprietary software. Accordingly, Lerner and Tirole give several reasons for why this should be the case. While discussing the immediate benefits they point out that producing OS software might be easier than the production of proprietary software because of the "*alumni effect*": the programming environment of an OS project might be familiar to the programmer because such environments are regularly used in schools and universities for learning purposes. They also mention the benefits which result from personal use: they point out that contributions like *customizing* an OS product or *fixing a bug* in it bring private benefits that contributions to proprietary software do not normally bring.[34]

Lerner and Tirole also mention several reasons why the signaling incentive is stronger in the case of OS software. They claim that OS projects provide a *better performance measurement* of programmers than ordinary commercially developed programs, because in an open source project outsiders can see exactly what the contribution of each programmer was. In addition, the availability of the source code enables outsiders to judge not only whether a programmer has produced a piece of software which "works" but also how difficult a task its production was.[35] To these claims one might, of course, wish to object that also many commercial corporations have introduced methods for measuring the quality of the work of their programmers. In addition, the advertising clauses of OS licenses which request the publication of the names of contributors or institutions have in recent years been experienced as problematic, in particular when OS software has been commercialized. E.g. the use of BSD style licenses might lead to a situation in which companies would have to add a large number of programmer names to their marketing material, which most commercial companies would for obvious reasons be unwilling to do.[36] This problem has been addressed by the University of California at Berkeley, which is the author the BSD license. Such advertising clauses have been removed from Berkeley style licenses in 1999.[37] This opens new opportunities for making investments in OS&OC, but weakens the performance measurement effect which Lerner and Tirole mention.

The second reason why OS projects provide better performance measurement according to Lerner and Tirole is that each OS programmer is "her own boss", unlike a programmer who works for a hierarchical commercial firm and whose performance depends on her supervisor's interference, advice and other similar factors. Finally, as a reason of a third kind Lerner and Tirole mention the fact that many elements of source code are shared across open source projects and that more of the knowledge that programmers have accumulated can be transferred from one OS project to another than it is the case in the context of proprietary

[33] Ibid., p. 214. In this context Lerner and Tirole refer to Holmström (1999), a paper which discusses a simple economic model of managerial incentives.
[34] Lerner – Tirole (2002), pp. 16-17.
[35] Ibid., p. 17.
[36] See e.g. the discussion of BSD-style licenses on the Internet portal of the Free Software Foundation (http://www.gnu.org/philosophy/bsd.html , accessed on August, 29, 2002).
[37] Ibid.

software projects.[38] In other words, the performance of a programmer within one OSS project usually tells more about what her performance within another OS project would be than it is the case when the two projects produce proprietary software.

Lerner and Tirole's account of the incentives to contribute to OS projects leads to the conclusion that the contributors to OS projects *either* derive direct benefits from their contributions, as they did in Johnson's model in subsection 3.1 above, *or* have strong signaling incentives, which might be incentives to signal one's talent to peers, prospective employers, or the venture capital community.[39] While discussing the question *what kind* of OS software these incentives bring programmers to produce Lerner and Tirole reach conclusions which much resemble those of Johnson. They point out that the tasks which will belong to OS projects and which would be important for the least sophisticated users, like e.g. writing documentation or designing easy-to-use interfaces, do not provide strong signaling incentives and that on the basis of their analysis one could expect that such tasks would not get completed.[40]

As empirical evidence which fits their analysis Lerner and Tirole mention the facts that user benefits have been essential in many OS projects, that giving credit to authors has been essential in the OS movement, and that the reputational benefits that contributors have received appear to have had real effects on them. E.g., the Apache project recognizes all its contributors on its web site, highlighting the most committed contributors, and many of its contributors have subsequently been hired by commercial companies such as IBM.[41]

Steven Weber discusses in Weber (2000) the process of OS software development in more general terms than Lerner and Tirole do in their microeconomic analysis, and we shall return to a discussion of Weber's views in the subsequent sections of this report. While discussing the microeconomic basis of the OS phenomenon Weber mentions the interest that contributors feel in the problems that they solve and the reputation that they achieve as motives of contributing to OS projects. In particular, he points out that also "drudgery-type", seemingly uninteresting tasks might get taken care of in OS projects, because within the large community of contributors there might be someone who does not experience such tasks as drudgery but as challenging and interesting, or someone for whom completing the task is important because of a job responsibility.[42]

When Weber discusses the significance of reputation and "ego-boosting", he presents some criticisms of Lerner and Tirole's account of these motives. As we already saw above, in Weber's view participation in an OS project might lead not only to "ego-boosting" but also to "ego-damage", since a contribution might also get rejected. However, the OS community can diminish the significance of this effect by carefully focusing its criticisms on the code itself rather than on the person who has produced it.[43] A more serious criticism is that reputational concerns cannot explain the *successful cooperation* within OS projects. There would, Weber claims, be more competition for the leading positions of OS projects than is currently observed if the primary motivation of the contributors was the wish to obtain a reputation. Such competition could either take the form of directly challenging the position of a project leader,

[38] Lerner – Tirole (2002), pp. 17-18.
[39] Ibid., p. 18.
[40] Ibid., p. 19.
[41] Ibid., p. 20.
[42] Weber (2000), p. 25.
[43] Ibid.

like e.g. the position of Linus Torvalds, or the form of “strategic forking”. By strategic forking Weber means the practice of forking a project not for technical reasons but simply in order to become the leader of the new project which the forking creates. However, according to Weber neither of these phenomena is observed in e.g. the case of Linux.[44]

3.3 OS Software production and Network Externalities

Above in the beginning of Section 3 we pointed out that, if the motive of writing OS software is personal use, the fact that such software is made available to other users on the Internet is in need of a separate explanation. One would not expect that the producer of the software would in this case be willing to distribute her work freely if there is similar software which has a monetary value, or if she works for a commercial company for whose competitors the software is useful. However, as we stated in the beginning of Section 3.1, in this respect the phenomenon of OS software is not quite unprecedented, since there have been *user innovator* communities also in other fields. However, it should be observed that in the field of OS software the significance of *manufacturers* is fundamentally different from the significance that manufacturers have for other user innovator communities. In other fields, like e.g. in the field of sports equipment, it is essential that the users reveal their innovations not just to other users but also to manufacturers, but this is not necessary in the field of OS software. Of course, also the innovations of OS software programmers can get revealed to both to manufacturers which are “hostile” to the OS community in the sense that they wish make use of them in their proprietary software and to ones who are “friendly” in the sense that they sell OS software and related services.[45] However, unlike in other fields, in the field of OS software an innovation of one user can be used by the others without the interference of manufacturers.

Following von Hippel the phenomenon of freely revealing innovations to others can be analyzed by weighing the costs of free revealing against the benefits from it. When the costs of revealing the innovation for free are low, even moderate benefits might be a sufficient reason for doing it. These benefits might include items which we have already considered, like reputational concerns.[46] The costs of revealing depend, of course, on the amount of rivalry between the adopters of the technology. Although such rivalry might be non-existent when the contributors to an OS project are motivated by the “hacker ideology” which we discussed in Section 3.2, there obviously must be such rivalry in other cases. Why do contributors nevertheless reveal their “innovations” – i.e. useful pieces of code – to their rivals? As Steven Weber points out, an obvious explanation is provided by the *positive network externalities* with which OS software is associated.[47]

By definition, a product is said exhibit *network externalities* or *network effects* it the value that it has to each user depends on how many other users of the same product there are.[48] Communication technologies provide the most obvious examples of products which exhibit network externalities: e.g. a *telephone* is the more valuable for its possessor the more there are other persons who have telephones and whom she can call with her telephone. The OS

[44] Ibid., p. 26.
[45] We shall discuss the latter possibility in some detail in subsection 5.2 below.
[46] Von Hippel (2001), pp. 85-86.
[47] Weber (2000), p. 28.
[48] See e.g. Shapiro - Varian (1999), p. 13.

products which are used for communicating with others are associated with network externalities in the same obvious sense. In order to see why also other OS products are associated with network externalities it is useful to draw a distinction between *direct* and *indirect* network effects.[49]

A direct network effect occurs in a situation in which there is a concrete network which has the considered products as its components and which is such that the value of the products which belong to the network depends on its size. The network effects which are associated with telephones or railroad tracks are in this sense direct. On the other hand, an indirect network effect occurs in a situation in which the popularity of a product *improves the availability of other products which are complementary to it*. For example, an *operating system* is not a product which is associated with direct network effects, since a person who has installed an operating system to her computer does not derive any direct utility from the fact that also others are using it. However, the popularity of an operating system is useful for its users in so far that its popularity tends to increase the amount of available software which is compatible with it. Hence, operating systems are associated with indirect network effects.

As Weber points out, in addition to such direct and indirect network effects there is one more important externality with which OS software is associated: the more users such software has, the more efficient is *debugging*. Hence, an OS project benefits from the free-riders who just use the software without making any modifications to it, at least as long as in addition to the free-riders there is also a 'core' group who takes care of programming.[50]

Weber also points out that the economic logic of OS projects is an inversion of the logic of both standard *intellectual property rights arguments* and the *theory of collective action*. In the intellectual property rights thought to which he refers the challenge is to create incentives for the innovator, but once an innovation is there, its distribution takes care of itself through the market. However, in the context of OS software the problem is the distribution - i.e. the problem is to provide the OSS product with sufficiently many users for making it seem worthwhile to contribute to it - but once this has been achieved, the innovation takes care of itself.[51] Similarly, the theory of collective action has largely been focused on the problems of motivating behavior which is collectively rational when such behavior does not seem rational from a narrowly egoistic perspective,[52] but Weber concludes that OS software is not so much a problem of collective action theory than a problem of *coordination*.[53] Below in Section 4 we shall shortly discuss the coordination problems of OS projects.

[49] This distinction is drawn in e.g. Economides (1996), p. 679.
[50] Weber (2000), p. 29.
[51] Ibid.
[52] In this context Weber refers to Hardin (1982). The so-called "prisoner's dilemma", a problem of motivating behavior which does not seem egoistically rational, has been given a prominent place in this introduction to the theory of collective action. (Cf. ibid., pp. 22-25.)
[53] Weber (2000), p. 29.

4. On the "Economic Logic" of the OS&OC Software Production Process

As it was stated above, in this section of our survey of economic perspectives on OS and OC software we shall consider the economic analyses of how individual OS projects function. Also their organization poses puzzling questions for the economist, just like the behavior of the individual programmers who contribute to them did. Successful OSS projects might have an extremely large number of contributors. According to a famous metaphor which has been suggested by *Eric S. Raymond* in the essay "The Cathedral and the Bazaar", the community of Linux developers resembles "a great babbling bazaar of differing agendas and approaches",[1] whereas a more centralized approach to software development in which individual programmers or small groups of them work in isolation resembles building a cathedral. The OS products which emerge as the result of the seemingly anarchistic and uncoordinated "bazaar-like" activities of a very large number of contributors are sometimes extremely complicated: for example, a full distribution of Linux contains more than 10 million lines of code.[2] It seems quite puzzling that developers who work in a parallel and relatively unstructured way manage to produce such large, complex, and useful systems of code, which can successfully compete with their commercial and – to use Raymond's metaphor – "cathedral-like", carefully planned alternatives.[3]

Steven Weber emphasizes that this success cannot be explained by simply referring to the 'self-organization' of OS projects, which he does not accept "as a placeholder for undisclosed microfoundations or social mechanisms".[4] Rather, as we stated in the Introduction, according to him an explanation of open source must contain three items: the 1) microfoundations – i.e. the motives of the individual programmers – 2) the "economic logic situating the collective good at play", and 3) the "social and political structures" which lead to successful OS products, given the individual motivations and the economic logic of software creation.[5] Economic aspects have an important role in such an explanation since, despite of the fact that the production of OS software has 'artistic' and creative elements, and that it is sometimes motivated by ideological views, it has become "deeply embedded in an economic setting".[6] Weber also daims that even if there actually was an OS project whose contributors were motivated by e.g. the beliefs that "make up an informal hacker culture" rather than by economic considerations, and even if such a project were successful, it would be *vulnerable to invasion*. Programmers who are motivated by other, more crass considerations might try to take such a project over.[7] This attempt might take the form of *strategic forking*; i.e., the project might get invaded by programmers who would try to fork it for strategic reasons, simply in order to become the leaders of the resulting new project. However, as we mentioned in subsection 3.2, it seems that the phenomenon of strategic forking is not observed.

[1] Raymond (2001), p. 21.
[2] Weber (2000), p. 33.
[3] Cf. ibid., pp. 3-4.
[4] Ibid., p. 35. For an account of OS software development which emphasizes the self-organizing qualities of OS projects, see Kuwabara (2000), in particular Chapter 5.
[5] Weber (2000), pp. 19-20.
[6] Ibid., p. 35. See also ibid., p. 22, where Weber criticizes Lerner and Tirole because they do not give any detailed account of the "macro-level organization" of an OS project but, according to Weber, retreat into a "notion of charisma" while discussing it.
[7] Ibid., p. 27.

According to standard organization theory, complexity in the division of labor tends to lead to formal organizational structures.[8] However, there are two obvious reasons which reduce the need for such structures in OS software projects: on the one hand, *the Internet* has dramatically lowered the communication costs of OS projects and, on the other hand, OS projects utilize *source code modularization.*[9] This means that large OS programs work by calling relatively small and self-contained modules which communicate with each other with clearly specified communication interfaces. When one succeeds in constructing software with these characteristics, it suffices for each contributor to have a detailed knowledge of only those modules that she has produced, and each contributor is also able to make improvements and corrections to her modules without requesting the other contributors to make corresponding changes to the other modules of the product. According to Weber, these facts "reduce organizational demands on the social/political structure" but do not, of course, make such demands disappear. Rather, the successful OS projects have a structured organization of authority: as we saw in subsection 2.2, e.g. the Linux project is led by Linus Torvalds and the Apache project is governed by a committee.[10]

Brooks' law constitutes an obvious problem for any theoretical account of the organization of OS projects. *Frederick P. Brooks* put forward this "law" while criticizing the idea that one could speed a software project up simply by hiring more programmers for working within it. He claimed that this would, in general, not be possible, since a larger work force would have to spend a larger part of its working time in communication than a smaller one. According to Brooks' law, "*adding manpower to a late software project makes it later*".[11] However, OS projects seem to be making efficient use of the work of an extremely large number of programmers, and one of the obvious tasks of any theoretical account of the organization of OS projects is to explain how this can be the case despite of Brooks' law. We shall discuss this question in Subsection 4.1 below. We then proceed to a more systematic discussion of the organization of OS projects. We begin by presenting in subsection 4.2 some key principles of organizing OS projects which OS programmers have themselves put forward and which have been summarized by Steven Weber. After this we move to a discussion of the more theoretical reflections on the same topic: in subsection 4.3 we discuss the management of OS projects in more general terms, and in subsections 4.4 and 4.5 we present some theoretical perspectives from which OS projects have been considered as wholes.

4.1 Circumventing Brooks' Law

Frederick P. Brooks motivated the claim which is now referred to as "Brooks' law" by criticizing the practice of using *man-month* as a unit of measurement of the size of a software development project. Brooks pointed out that a man-month is a suitable unit for measuring the size of a task only when it *can be partitioned* between workers and *no communication* is needed between them.[12] In this case persons and months are *interchangeable* in the sense the

[8] See e.g. Mintzberg (1979), p. 18ff.
[9] Weber (2000), pp. 33-34.
[10] Ibid., p. 34.
[11] Brooks (1975), p. 25.
[12] Ibid., p. 16.

amount of person-months that it takes to complete the task is independent of how many workers participate in it. Figure 4.1 shows the amount of time that is needed for completing the task as a function of the number of persons who have been hired for working on it in this case.[13]

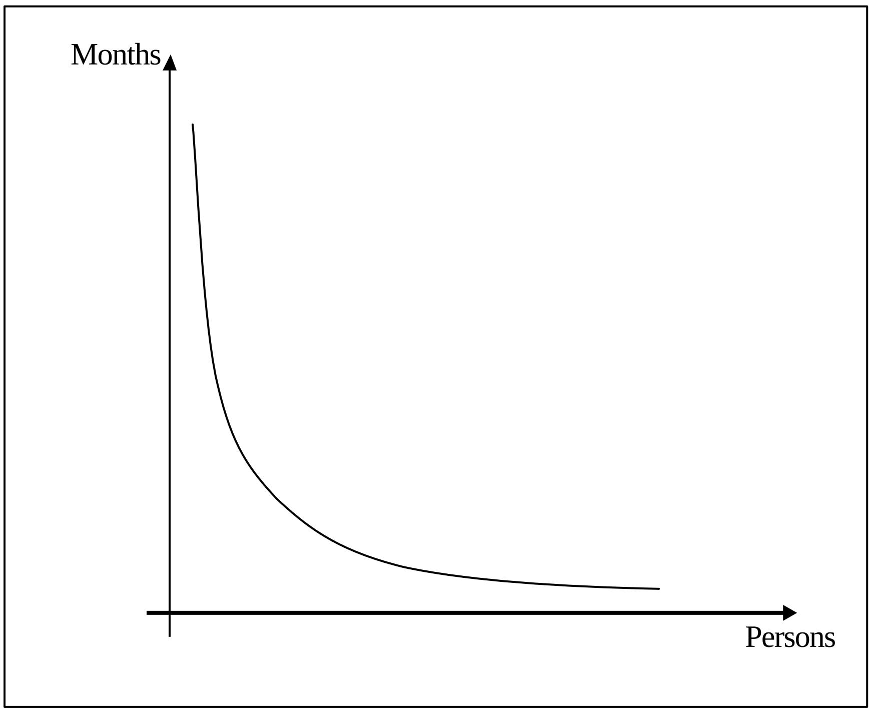

Figure 4.1. Time versus the number of workers in a partitionable task.

However, the situation will be different from the one represented in Figure 4.1 when there are *sequential constraints*, i.e. when some of the subtasks which the task contains can only be performed when some other subtasks have been completed. In the extreme case in which the task consists of parts which must be completed in some fixed order and each of which must be completed by a single person, the amount of time that it takes to complete the task is independent of the number of persons that are working on it. Figure 4.2 represents a situation of this kind. Brooks claims that in the field of software development many tasks are of this kind "because of the sequential nature of debugging", i.e. because debugging can only be started when a preliminary version of the debugged software is available.[14]

[13] Cf. ibid., p. 16..
[14] Ibid., p. 17..

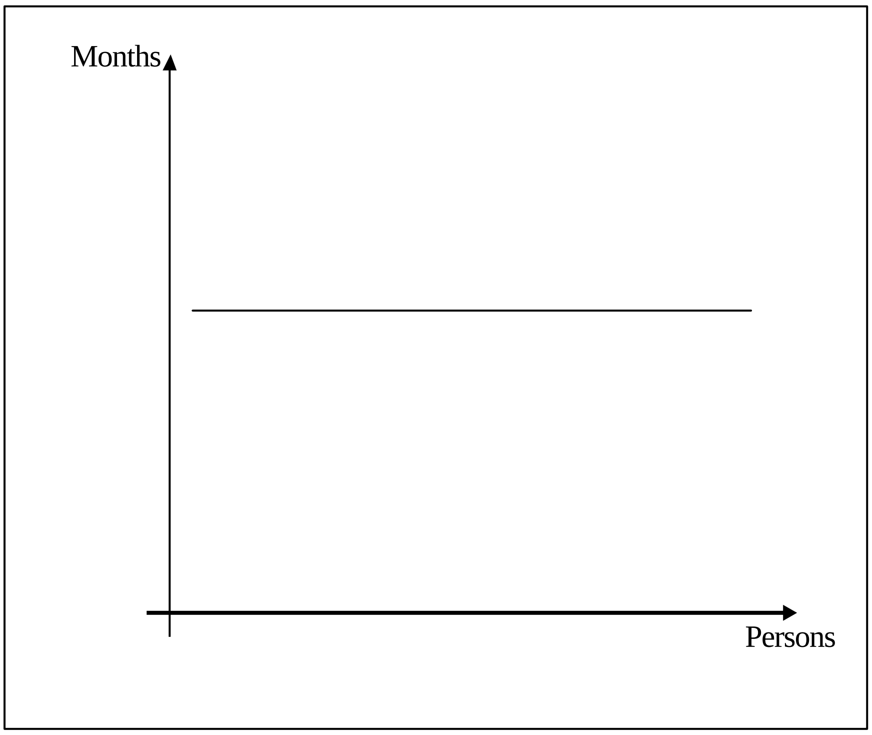

Figure 4.2. Time versus the number of workers in a unpartitionable task.

Further delays are caused by the necessity of *communication* between workers. Brooks divides the "burden of communication" into two parts, *training* and *intercommunication*. On the one hand, workers must be trained for their jobs. This is a task which cannot be partitioned, and the total number of man-months spent in it is proportional to the number of workers that are employed by the project. According to Brooks, the total time spent in intercommunication between workers grows even more when their number is increased: if each worker must communicate with every other worker, the amount of time spent in intercommunication is proportional to the number of different *pairs* of workers.[15] When the number of workers is n, the number of such pairs is $n(n-1)/2$. This means that if each of the workers has to spend some fixed amount of time in communicating with each of the other workers, the time that gets spent in communication is approximately proportional to the *square* of the number of workers who are involved in the project.

These effects have the consequence that the amount of time which is needed for completing the task might actually *grow* when more workers are employed for completing it. A situation of this kind has been represented in Figure 4.3. As we mentioned above, according to Brooks a situation in which more workers are hired to a software project and in which this has the consequence of *delaying* the project might emerge when managers have underestimated the time which is needed for the project and try to speed it up by increasing the available work

[15] Ibid., p. 18.

force. Brooks claims that such an attempt will not be successful, and that *adding manpower to a late software project makes it later,*[16] which is the claim that is known as "Brooks' law".

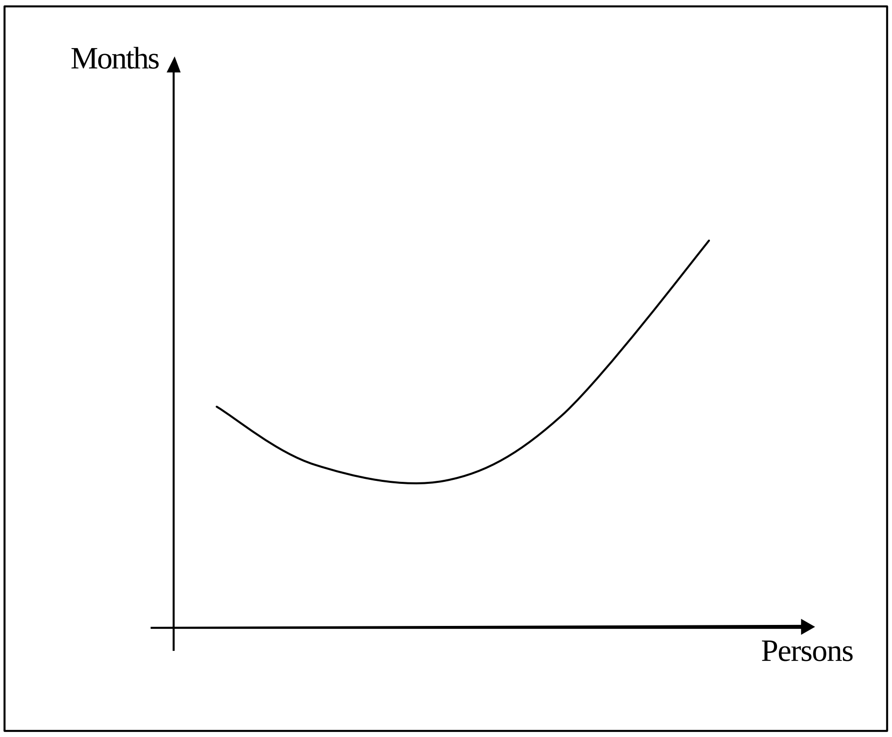

Figure 4.3. Time versus the number of workers in a case to which Brooks's law applies.

The "bazaar-like" OS projects successfully produce software products which have an extremely large number of co-authors. How can this be the case despite of Brooks' law? Two obvious answers to this question were already mentioned above: the *Internet* has dramatically lowered the costs of communication after Brooks put forward his law, and – what is more important – large OS products have a well-defined *modular structure.* Eric S. Raymond addresses in *The Cathedral and the Bazaar* the question how OS projects circumvent Brooks' law, and he points out that "Brooks's Law analysis (and the resulting fear of large numbers in development groups) rests on a hidden assumption", namely that "everybody talks to everybody else".[17] However, in Raymond's account the persons involved in an OS project are divided into the "core group", which is quite small and often contains from one to three persons, and the "halo". According to him the "halo developers work on what are in effect separable parallel subtasks and interact with each other very little" and it is only within the small core group in which "we pay the full Brooksian overhead".[18]

[16] Ibid., p. 25.
[17] Raymond (2001), p. 35.
[18] Ibid., pp. 34-35. Cf. Weber (2000), pp. 33-34.

In addition, Raymond points out that OS projects are utilizing a very *efficient debugging technique*. OS projects release software often and at an early stage of its development, and utilize the error reports that users produce.[19] These error reports are often more informative than the ones that commercial software developers get from their customers, because they are often *source-code-level bug reports*: the user of an OS product might not just report in which way the software has behaved unsatisfactorily, but also study the relevant source code and make a suggestion concerning the precise nature of the mistake in it.[20]

We pointed out already in Section 3.3 above that such error reports turn free riding – i.e. the practice of just using the software without developing it further – into a positive thing for the OS project. Such effects could, in principle, exist even if the users and developers communicated with each other without using the Internet, like by sending code on CD's. However, Weber claims that the Internet changes communication much more dramatically than by just speeding it up. According to him the "massive bandwith of Internet communication is a change in kind, not just in degree": it enables the open source community to engage in *distributed innovation* and almost parallel processing of complex tasks.[21]

Raymond claims that when the project has sufficiently many "beta-testers" who are willing contribute to it in this way, almost every problem will get characterized quickly, and that also the solution of almost every problem will be obvious to someone. This means that – contrarily to Brooks' law – there is a way of parallelizing debugging efficiently. Raymond summarizes this point by stating that "Given enough eyeballs, all bugs are shallow", a claim to which he refers as "Linus's law".[22]

4.2 Some Typical Features of Successful OS Software Development

Steven Weber has presented a list of what he calls "key principles for what people do in Open Source". For a large part these principles are summarized versions of ideas which are contained also in Raymond's essay *The Cathedral and the Bazaar*, to which we have already repeatedly referred. According to Weber, the principles that he puts forward are based, in addition to Raymond's analysis, also on a set of interviews.[23] In this section we shall have a closer look at these principles.

The first of Weber's principles is: "*Pick Important Problems and Make Them Interesting*". According to Weber user-developers are willing to contribute to projects which they believe will produce important additions to software, or at least products which are "cool", i.e. products which have new functions or do things in a particularly elegant way.[24] The second principle is: " '*Scratch an itch*' ". By this Weber means that individual contributors to the project are willing to make contributions which solve problems that are important for them personally. This implies that it is quite essential for an OS project that it has a sufficient number of contributors: when the number of contributors is sufficiently large, most problems

[19] Raymond (2001), pp. 28-29.
[20] Ibid., p. 35.
[21] Weber (2000), p. 36.
[22] Raymond (2001), p. 30.
[23] Weber (2000), p. 15.
[24] Ibid.

that must be solved will be such that there is someone in the community who is affected by the problem personally, and who is accordingly willing to try to solve it.[25]

As his third principle Weber states: "*Reuse whatever you can*". He points out that, for two obvious reasons, the OS programmers have a stronger incentive for not "reinventing the wheel", i.e. for not writing software which performs identical functions with some already existing software, than the producers of proprietary software. Firstly, the producers of OS software do not receive monetary compensation for their work, which the producers of proprietary software receive also when their work does not utilize the previously written software efficiently. Secondly, unlike proprietary software producers, OS producers do not have to worry about questions of copyright, but can know for certain that they have and will always have the right to make use of the previously existing source code.[26]

The fourth principle on Weber's list is: "*Use a parallel process to solve problems*".[27] As it was already stated in subsection 4.1 above, OS projects use a parallel process in both debugging and innovation. In OS projects the users of the software take care of a large part of the necessary debugging, since programs are released at an early stage of their development and their developers make use of the error reports that the users of the programs produce instead of trying to remove all bugs from the software before its release. The parallel process which is used for making innovations, i.e. for the inclusion of new features in the software, is such that many authors propose solutions to the same problems and write software which implements similar features. Given their proposals, the project will be able to choose the best of the alternatives which are presented to it. This is, in a sense, wasteful, but at the same time it resembles closely what happens in an evolutionary process in nature.[28]

Weber's fourth principle is closely linked with the fifth one, which is: "Leverage sheer numbers". By this he wants to make the same point which we above expressed by the so-called Linus's law, "Given enough eyeballs, all bugs are shallow": if the software is used by sufficiently many users, some of its users will run into each of the bugs that it contains.[29]

Weber's sixth principle is "*Document well*". He states that, although documentation is "a time consuming and sometimes boring process", it is essential for an OS project: it is essential that the programmers make their intentions clear, so the contributors who utilize or modify their work later on can understand which functions the particular pieces of code have in the larger scheme.[30] The seventh principle is: " *'Release early, release often'* ". The parallel process of problem solution which we discussed above is based on the idea that, instead of trying to find all the bugs in the software before releasing it, software developers leave it to the users of the software to find a large part of the bugs and to report about them. This is a very efficient method of finding bugs, and it is a method of which OS developers can make much more use than the developers of proprietary software, because the customers who pay for their software would be dissatisfied with early and "buggy" releases of software, but the user-developers of OS software have quite different expectations.[31]

[25] Ibid., pp. 15-16. Cf. Raymond (2001), pp. 23-24.
[26] Weber (2000), p. 16. Cf. Raymond (2001), pp. 24-25.
[27] Weber (2000), p. 16.
[28] Ibid. Cf. Raymond (2001), p. 30.
[29] Weber (2000), p. 16. Cf. Raymond (2001), p. 30.
[30] Weber (2000), p. 17.
[31] Ibid. Cf. Raymond (2001), p. 29.

4.3 The Management of OS Projects

As it was stated above, the contents of the seven principles which we considered above are, for a large part, contained in a somewhat longer list of aphorisms with which the OS programmer Eric S. Raymond has himself characterized the field in which he is working. Next we shall discuss the management of OS projects in a more systematic manner. In this discussion we shall lean heavily on an important recent paper by Josh Lerner and Jean Tirole, Lerner – Tirole (2002), and on the work of Steven Weber. Lerner and Tirole emphasize that, despite of the seemingly anarchistic features of an OS project, the quality of its *leadership* is quite essential for its success. In particular, it is quite essential for an OS project that the right to modify the official version of the software is restricted. Accordingly, in an OS project this right is normally confined to a committee, as it is the case in the Apache project, or to a single leader, like e.g. to Torvald in the case of Linux. [32] The leader or the leadership of the project has several functions. Lerner and Tirole state that the leader must "provide a 'vision', attract other programmers, and, last but not least, 'keep the project together' (prevent it from forking or being abandoned)".[33]

In order to get the project off the ground the leader must, obviously, initially produce a sufficient amount of code to make the project seem worthy of contributing to. According to Lerner and Tirole, the part of the work which is completed by the leader originally must, however, not be too large: rather, other programmers will be motivated to join the project only if they are given the impression that challenging programming problems have been left to them.[34] Obviously, one of the problems in attracting programmers at the early stages of the project is the fact that the quality of its leaders is then still untested. However, as Lerner and Tirole point out, at the early stages of the project programmers can be motivated to make contributions by the fact that if the project turns out to be a success, early contributions to it will be very visible. Lerner and Tirole exemplify this point by discussing the initial version of Linux, which was quite minimal, and in which Linus Torvalds sought to create interest by emphasizing the amount of creative programming that would be needed in order to make it fully functional. [35]

A *charismatic* or *trusted leader* is, according to Lerner and Tirole, quite essential for an OS project also in so far that such a leader will "substantially reduce the probability of forking" in two ways. On the one hand, if there is a disagreement concerning the way the project should be developed, the indecisive programmers are likely to support the alternative preferred by the leader, and, on the other hand, the dissenting factor may fail to have a similar obvious leader of its own.[36] In his discussion of the reasons what stops OS projects from forking Weber emphasizes other factors. In particular, he refers to the *network externalities* which we considered already in subsection 3.3. It is in the interest of an OS programmer that she contributes to a large project, and it is difficult to convince programmers that a potential forker "could accumulate a bigger and better community of developers than already exists in the main

[32] Lerner – Tirole (2002), p. 221.
[33] Ibid., p. 220.
[34] Ibid.
[35] Ibid., p. 221.
[36] Ibid., pp. 222. Lerner and Tirole also mention some other reasons which stop OS projects from forking. These include the loss of economies of scale which would be due to the creation of smaller communities, the "hesitations of programmers in complementary segments to port to multiple versions", and "the stigma attached to the existence of conflict" (ibid.).

code base". [37] In addition, *reputation* depends on the size of audience, and the audience would be split by forking. Accordingly, Weber states that open source "creates a mildly counterintuitive dynamic: the more open a project is and the larger the existing community of developers, the less tendency to fork". [38] Accordingly, OS production is an "ecosystem" with "winner-take-all dynamic": it is difficult to start an alternative project in "ecological niches" which are already occupied. [39]

Just like Lerner and Tirole, also Weber emphasizes the significance of *competent leadership* for an OS project. As an example he considers Linus Torvalds, who invests "huge effort in maintaining his reputation as a fair, capable, and thoughtful manager" [40] by e.g. justifying his decisions to the Linux community in a detailed manner. As Weber states, many in the Linux community believe that a leader of a worse quality would have either undermined the Linux community altogether or made the Linux developers exit and create a new community. [41]

Weber discusses also *conflict resolution within OS&OC communities* in some detail. He refers to Raymond, who has divided the decisions concerning which conflicts emerge in OS projects into disagreements on the *design* of the software (i.e. on which pieces of code to include), disagreements on the persons who get *credited* for different contributions and on how they are credited, and disagreements concerning the *prevention of forking*.[42] According to Weber, the resolution of conflicts in OS communities is based on the one hand on *cultural norms* and on the other hand on more objective *technical criteria*.[43] OS projects have also *sanctioning mechanisms* against contributors who violate their norms. These include, according to Weber, *'flaming'* by which he means " 'public' condemnation (usually via e-mail lists) of people who violate norms" and *'shunning'*, i.e. refusing cooperation. [44]

The cultural norms of OS communities include the idea that *authority follows and derives from responsibility* – i.e. the idea that the authority that a person has in the community depends on the amount of responsibility that he or she takes for pieces software – and *seniority rules*. By seniority rules Weber means the idea, stated by e.g. Eric S. Raymond, that when there is a conflict between two contributors or groups which cannot be resolved objectively, the side who has put more effort into the project wins it. [45] Also the more objective, technical criteria exist only inside a *cultural frame*. In the case of the OS projects which have produced variants of the Unix operating system this cultural frame is based on the shared experience in Unix programming but, according to Weber, the cultural frames are nevertheless different in such projects: e.g. Open Source Initiative represents a pragmatic view in which technical excellence is central, whereas the cultural frame of Free Software Foundation is more ideological. [46]

Weber also claims that *property rights* or *ownership customs* must be a key part of the macroeconomics of open source.[47] While discussing the way in which property rights of an OS project are obtained Weber quotes Raymond, who mentions three ways of obtaining them.

[37] Weber (2000), p. 30.
[38] Ibid.
[39] Ibid., p. 32.
[40] Ibid.
[41] Ibid.
[42] Raymond (2001), p. 100. (Cf. Weber, 2000, p. 18.)
[43] Weber (2000), pp. 30-31.
[44] Ibid., p. 32.
[45] Ibid., p. 30-31. (Cf. Raymond, 2001, p. 103.)
[46] Weber (2000), p. 31.
[47] Ibid., p. 24.

The first of these is to found the project, the second one is to have the project handed to oneself by the previous owner, and third one is to pick up a project whose owner has disappeared or lost interest in it.[48] Such property rights are important despite of the fact that open source licenses allow anyone to make modifications to open source software and to distribute them: according to Raymond there is a well-recognized distinction between "'official' patches" which have been approved by the publicly recognized managers of the project, and "'rogue' patches" which have not been approved in this way, and which are "unusual, and generally not trusted".[49] According to Weber it is difficult to give a satisfactory *economic explanation* for the ownership customs which exist in the field of OS production, because it is difficult to evaluate the returns that one gets from owning an OS project. Nevertheless, it is clear that the property rights to open source projects are important: their importance is demonstrated by the "time, energy and emotion" which is put into defending them when the norms concerned with their ownership are breached.[50]

Yet another interesting question which Weber addresses is the effect that the *necessity of communicating with other institutions* has on the organizational structures of OS projects. As Weber points out, institutions which communicate with each other tend to build similar organizational structures.[51] When this idea is applied to OS projects, it means that, as the popularity of OS products grows and commercial companies start using them more, OS projects may wish to build organizational structures which resemble those of the commercial companies with which they have to communicate. For example, within the Linux community there have been views that it was important to create "an impression of organizational credibility for open source" which would "appeal to and reassure commercial users".[52] This aim is promoted by e.g. the non-profit corporation *Open Source Initiative*, whose aim is to make OS software more attractive to the commercial world.[53] On the other hand, it might also happen that a commercial company which cooperates with OS projects might introduce structures which resemble those of an OS project. The development of Netscape, a commercial software producer which has started an OS project of its own, might be quoted as an example of such a case. We shall shortly discuss this project in subsection 5.2 below.

Ruben van Wendel de Joode and *Jeroen Kemp* have addressed the problems of the management of an OS project on a somewhat more general level of abstraction. They have discussed the *strategy finding task* of an OS project, which they view as analogous with the strategy finding tasks of ordinary firms.[54] According to them the literature on strategy finding often represents either of two opposite views: it either represents the *market-based view of the firm*, in which the firm's strategy finding consists mostly of achieving a competitive position on the market via e.g. cost-effectiveness, or the *resource-based view of the firm*, in which the success of the firm depends on those of its internal capabilities which are specific for it.[55] Their own account of strategy finding contains elements from both views. They present a list of key aspects of the strategy finding task of an organizational system, which contains the

[48] Raymond (2001), pp. 74-75. According to Raymond these norms resemble the norms concerning *land tenure* which are inherent in Anglo-American common law. (Ibid., pp. 76-77.)
[49] Ibid., p. 74.
[50] Weber (2000), p. 24.
[51] Ibid. Cf. DiMaggio – Powell (1983).
[52] Weber (2000), p. 34.
[53] Cf. the internet portal of the Open Source Initiative, http://opensource.org.
[54] Van Wendel de Joode – Kemp (2002).
[55] Ibid., p. 4.

following items: *1) purpose* (i.e. the purpose at which the organization aims), *2) value proposition* (i.e. specification of the "kind of value" that the organization offers), *3) external possibilities*, and *4) internal capabilities*.[56]

Viewed from this perspective, an essential feature of the OS projects is their flexibility, i.e. their ability to change the other aspects of their strategy finding tasks as the external possibilities change. For example, as van Wendel de Joode and Jeroen Kemp point out, Linux has dramatically changed both its "purpose" and its "value proposition" as its external possibilities have changed. They present a classification in which the history of Linux is divided into an "initial stage", a "growth stage", and a "maturity stage", and according to them the "value proposition" has changed from the "basic product" in the initial stage to the "basic product with numerous applications" in the growth stage, and it has included also "services and training" in the maturity stage. Similarly, the "purpose" of the Linux project has changed in the course of its development: according to van Wendel de Joode and Jeroen Kemp the "purpose" of Linux was originally, in the initially stage, "intellectual interest", in the growth stage "usability" and in the maturity stage "user-friendliness".[57] The user-friendliness of Linux is, obviously, a particularly important aim for the commercial companies like Red Hat or IBM which produce products that are complementary to Linux.[58]

4.4 Some Analogies of OS Projects

Both the members of the OS communities and the outsiders who have reflected on such communities have discussed their characteristic features also on a more general level of abstraction. For example, Eric S. Raymond has put forward the idea that OS communities are what anthropologists have called *gift cultures*.[59] According to him, a gift culture is not an adaptation to scarcity, like e.g. an exchange economy is, but an adaptation to *abundance*. In a gift culture one's social status does not depend on what one controls but on what *one gives away*. Gift cultures are, according to Raymond, to be found among some aboriginals "living in ecozones with mild climates and abundant food" and they are also exemplified by some behavior of the "very wealthy" in our own society, like e.g. by the "multimillionaire's elaborate and usually public acts of philanthropy".[60] Raymond claims that the subculture of the hackers is a gift culture in a similar sense:[61]

> …it is quite clear that the society of open-source hackers is in fact a gift culture. Within it, there is no serious shortage of the 'survival necessities' – disk space, network bandwidth, computing power. Software is freely shared. This abundance creates a situation in which the only available measure of competitive success is reputation among one's peers.

[56] Ibid., p. 5.
[57] Ibid., p. 6.
[58] Ibid., p. 7.
[59] Raymond (2001), pp. 80-82.
[60] Ibid., p. 81.
[61] Ibid.

However, Steven Weber has criticized this way of looking at OS software production. Weber claims that there is a flaw in Raymond's argument. This flaw has, according to Weber, to do with the notion of *abundance* that Raymond is using. Although e.g. computing power and disk space are abundant resources in the 'ecosystem' of OS programmers, the "time and brainspace of smart, creative people" are not. The situation is analogous with that in the field of literature, in which abundance of paper "does not translate directly into an abundance of great writing".[62] In other words, what the OS programmers are giving away for free are non-abundant resources, and the idea that gift cultures emerge in circumstances of abundance does not adequately explain their behavior.

A more conventional analogy of OS software production is constituted by ordinary *scientific research*, in which researchers make the results of their work freely available to anyone.[63] Above in subsection 3.1 we mentioned another obvious analogy, the *user innovation communities* which have already before the emergence of OS software existed in other fields, like e.g. in the field of developing sports equipment or of developing scientific instruments.[64] A closely related notion is that of a *horizontal innovation network*. Eric von Hippel uses this expression for referring to a network of the users of a good of a specific kind. He draws a distinction between innovation "manufacturers" who "expect to profit from an innovation by selling it in the marketplace" and innovation "users" who expect to "profit from an innovation by in-house use".[65] An innovation network which is horizontal in his sense emerges when there is no specific group of innovation "manufacturers". In such a network innovative products are built for personal use, and innovations are freely revealed to others.[66] Clearly, OS projects fit von Hippel's characterization of horizontal innovation networks fairly well.[67] According to von Hippel, "user-only innovation development, production, distribution and consumption networks" can flourish when three conditions are valid. These are:[68]

> (1) at least some users have sufficient incentive to innovate, (2) at least some users have an incentive to voluntarily reveal their innovations, and (3) diffusion of innovations by users is low cost and can compete with commercial production and distribution."

Above in subsections 3.1 and 3.2 we discussed at length the reasons why the conditions (1) and (2) are valid for OS projects, and since the software that OS projects produce is distributed over the Internet, also condition (3) is obviously valid for them.

---

[62] Weber (2000), p. 23.

[63] Cf. Kelty (2001). It has also sometimes been suggested that the more general framework of the theory of *epistemic communities*, within which scientific communities are viewed as epistemic communities of a particular kind, could be applies to OS production in a fruitful fashion. See Edwards (2001).

[64] Cf. von Hippel (2001), pp. 82-83.

[65] Cf. von Hippel (2002), p. 2.

[66] Ibid., p. 3.

[67] To be precise, they fit von Hippel's description except for the fact von Hippel's characterization of innovation users (as opposed to manufacturers) does not appropriately characterize the motives of many OS producers, who – as we saw in Section 3 above – might have motives which are distinct from *both* profit *and* the utility derived from own use.

[68] Von Hippel (2002), p. 3.

4.5 Some More General Theoretical Perspectives

Before moving to the discussion of the effects of OS software on the software market in Section 5, we shall still have a look at two more general economic perspectives on OS projects. *Egon Franck* and *Carola Jungwirth* view in Franck – Jungwirth (2002) OS projects from the perspective of the familiar distinction between nonprofit organizations, which are financed by donations, and ordinary commercial firms. Franck and Jungwirth point out that, in general, when there are people who are willing to contribute to the production of a public good in the form of making donations – like e.g. the persons who make donations to a *listener-sponsored radio station* are – a nonprofit form of organization is better suited for collecting the donations than a firm which aims at making a profit. The organizations of both forms are faced with free-rider problems but, unlike a non-profit organization, the commercial firm is faced with an additional problem: the potential contributors might suspect that their contributions will be distributed to the owners of the firm instead of being used for the purpose for which the donation was given.[69]

An OS project resembles a non-profit organization in so far that, just like there are legal constraints which prohibit non-profit organizations from distributing their profits to those who are in control of them, the software produced by an OS project is protected by a license which stops the leaders of the project from turning it into their private property. At the same time, as we saw in the previous section, an OS project serves also aims which are analogous with the aim of making profit: the persons who contribute to it might be motivated by e.g. the wish to become famous or the wish to improve their employment opportunities. Franck and Jungwirth formulate this point by stating that OS projects have a "governance structure" which "enables investment without crowding out donations".[70] In other words, it is quite possible to "invest" in an OS project in the sense of making a contribution in order to receive a "profit" – e.g. in order to improve one's employment opportunities – but, unlike in profit-producing organizations of other kinds, this does not stop other people from making "donations" to OS projects for e.g. ideological reasons. The crafting of this "governance structure" is, according to Franck and Jungwirth, "basic institutional innovation in open source".[71]

Somewhat similarly, Eric von Hippel and Georg von Krogh have viewed OS projects as combining features from two models of encouraging innovation. They claim that presently "there are two major models characterizing how this may be done". The first of these is the "private investment" model, according to which "innovation will be supported by private investment".[72] In this model the free revealing of innovations will reduce the investors' profits, and it is in accordance with this model that society uses e.g. patents and copyrights for granting innovators rights which assist them in getting private returns for their innovation-related investments and which "increase innovators' incentives to invest in the creation of new knowledge".[73]

The other model which von Hippel and von Krogh have in mind is the "collective action model", which applies to a situation in which the produced innovation is a public good. In this model the contributors "relinquish control of knowledge they have developed for a project"

69 Franck – Jungwirth (2002), p. 17.
70 Ibid., p. 2. (Cf. p. 19.)
71 Ibid., p. 2.
72 Von Hippel – von Krogh (2002), p. 10.
73 Ibid.

and “make it a public good by unconditionally supplying it to a ‘common pool’”.[74] Under these circumstances it is, obviously, a problem to motivate the users of the public good to contribute to its production. This problem has had a central place in the literature on collective action. The ways of dealing with it which have been suggested in the earlier literature and which von Hippel and von Krogh mention include e.g. effective methods of recruiting participants and creation and deployment of selective incentives (e.g. bestowing credentials to the persons who are willing to contribute).[75] However, von Hippel and von Krogh point out that, although the participants of OS projects voluntarily contribute to the production of public goods, such projects “do not appear to follow any of the guidelines for successful collective action projects” that they describe and that have been proposed in the earlier literature.[76] For example, such projects make no active efforts to recruit contributors besides posting their goals and access address on a public website. The explanation for this situation has already discussed in Section 3: OS products are not pure public goods, since contributing to them may bring considerable private benefits also to the contributor.

Von Hippel and von Krogh claim that OS projects exemplify a model of innovation which is distinct from both the private investment model and the collective action model. They refer to this model as a “*private-collective model of innovation*”. This model combines some features of the two earlier models while eliminating some others. The new model differs from the collective action models in so far that it is not assumed in it that a free-rider could obtain from the public good benefits which equaled those that a contributor obtains.[77] On the other hand, it differs also from the private investment model in so far that it is not assumed in it that it would be unprofitable for an innovator to reveal his innovations: rather, in it “free revealing can increase innovation diffusion and so increase an innovator’s innovation-related profits through network effects”.[78]

[74] Ibid., pp. 10-11.
[75] Ibid., pp. 14-15.
[76] Ibid., p. 15.
[77] Ibid., p. 16-17.
[78] Ibid., p. 16.

5. Macroeconomic Aspects of Open Source Software Production

As it was explained above in the Introduction, our survey of economic perspectives on OS software is divided into three parts. In Section 3 we discussed the microeconomic explanations of the motives of the individual programmers who contribute to OS projects, and in Section 4 we considered the accounts that economists have given for the organization of OS projects. In this section we shall consider the topic of OS software from a *macroeconomic* perspective: we shall address the question how the emergence of OS software will influence the software market.

Until now this macroeconomic perspective has received an essentially smaller amount of attention in the relevant literature than the microeconomic perspective of Section 3 or the perspective of Section 4 which is, in a sense, located between the micro- and macroeconomic levels. Accordingly, below we shall rest content with discussing in rather general terms the relationship between commercial software manufacturers and open source software. Two obvious features of the software market are that it is a market with strong *network externalities* and that a very large part of the costs of software production belong to the category of *research and development costs,* since the production of copies of already existing software is almost costless. Below in subsection 5.1 we shall first quickly review some implications that these facts have for the competition in the software market, of which the competition between OS software and proprietary software is a special case. In subsection 5.2 we move to a more detailed discussion of the ways in which commercial companies could react to OS software, and in subsection 5.3 we shortly address the question how the emergence of OS software affects software *quality*.

5.1 Some Features of the Software Market

In Section 3.3 above we already shortly discussed the fact that software products are associated with network effects, since the value of a software product for its user often depends on the number of the other users that the product has.[1] We exemplified this fact by stating that e.g. the value of an operating system for its users depends on the number of its users, because the popularity of an operating system increases the amount of available software which is compatible with it. Networks effects are *demand-side economies of scale*, and just like the more familiar *supply-side economies of scale* – i.e., reasons which make the unit cost of a product grow smaller as it is produced in larger quantities – network effects cause *positive and negative feedback phenomena*.[2] For example, the popularity of a software product makes it more attractive to those who do not have it and tends to increase its popularity further, and conversely the unpopularity of a software product tends to make it even less popular.

However, the two kinds of economies of scale differ from each other in essential ways. *Carl Shapiro* and *Hal R. Varian* have illustrated their differences by contrasting Microsoft with

[1] Shapiro- Varian (1999), p. 13.
[2] Cf. ibid., pp. 175-180.

General Motors.[3] In the 20th century General Motors did not obtain in the United States any such monopolistic position as a car manufacturer as Microsoft currently has as a producer of operating systems of personal computers in the whole world. This, according to Shapiro and Varian, was because the economies of scale which promoted the growth of General Motors in the 20th century were supply-side economies of scale, and they got exhausted much earlier than would have been needed for creating a monopoly. However, the demand-side economies of scale do not dissipate in a similar way as the size of a company grows: the popularity of the products of Microsoft promotes their popularity further even in the current situation in the software market.[4] Accordingly, Shapiro and Varian claim that the *industrial economy* was populated by *oligopolies* – this has been the case in e.g. automobile, steel, aluminum, and petroleum industry – but that information economy is populated by *temporary monopolies*.[5]

As Shapiro and Varian point out, industrial and information economy differ from each other also in so far that customers' *expectations* concerning the success of a product have in the information economy a significance which they do not have in the industrial economy. This is because of the fact that when the value that a product has for its users depends essentially on how many other users it has, customers will be motivated to start using it only if they believe that other customers will do that as well.[6] In the context of our current topic this means that e.g. the popularity of Linux essentially depends on whether personal computer users can be made to believe that its popularity will grow in the future.

In the field of information technology a very large part of the production costs are due to *research and development*. When software production is viewed as an instance of research and development, OS software production can be taken to be an example of *non-proprietary innovation*, in which the innovator does not claim property rights to her innovation, whereas proprietary software production can be taken to exemplify ordinary *proprietary innovation* in which the innovator utilizes her innovation financially. In this report we shall not consider the economics of research and development in a detailed manner. Rather, we shall rest content by making a few remarks on the welfare effects of OS software.

Gilles Saint-Paul has discussed in general terms the topic of non-proprietary invention in Saint-Paul (2002), in which he argues that non-proprietary innovation "is not necessarily a free-lunch"[7] and presents an economic model in which the existence of non-proprietary invention reduces incentives to innovate in the proprietary sector. If this idea is applied to software production, it means that the availability of OS software would tend to lower the quality of proprietary software and the growth rate of software industry. However, such a conclusion could be criticized by pointing out that there are examples in which the competition between OS software and proprietary software clearly seems to have made the quality of proprietary software higher. E.g., the the competition between Netscape Communicator, which has become OS software, and Microsoft Internet Explorer, which is a part of the proprietary Microsoft Windows operating system, clearly seems to have made the quality of *both* products rise.

Saint-Paul also presents three additional reasons which are not included in his model and which can be expected to reduce the utility derived from non-proprietary invention further.

[3] Ibid., pp. 179-180.
[4] Ibid.
[5] Ibid., p. 173.
[6] Ibid., pp. 14-15.
[7] Saint-Paul (2002), p. 2.

The reasons which he mentions are that the design of non-proprietary goods has been "intended to fit the desires of the inventor" rather than those of the customers, that non-proprietary inventions "steal business" from the proprietary sector, and that "philanthropic innovation diverts talents" from the proprietary sector.[8] It seems that only the first of these claims supports the claim that non-proprietary innovation would decrease *total* welfare: when this claim is applied to the software market, it turns out to mean that commercial software producers would be more efficient in producing software that suits the users who cannot or do not want to contribute to OS projects themselves.

On the other hand, the welfare effects of *proprietary innovation* – like the production of proprietary software – can, in principle, be discussed by comparing the social welfare which results from an innovation with the profit which it brings to the innovator. A situation in which these were always equal would be socially optimal in the sense that in this case a profit-maximizing firm would be motivated to make an innovation precisely in those cases in which making it would increase welfare. However, according to standard economic theory there are several reasons why this is normally not the case. Three such reasons are mentioned in the recent paper Schmidt – Schnitzer (2002). Firstly, if the firm charges the same price from all of its customers, this price cannot measure the addition in the welfare of each of them. Secondly, the new product will reduce the profits of the producers of the old products which reduces welfare, and, thirdly, there will be technological spillovers to other firms which increase welfare, and neither of these effects is reflected in the profit of the innovator.[9] Accordingly, standard economic theory seems to yield no easy ways of evaluating the welfare effects of the emergence of open source software. As already stated, we shall nevertheless shortly return to the economic analysis of one aspect of its welfare effects, its effects on *software quality*, in subsection 5.3 below.

5.2 OS Software and the Market of Proprietary Software

As we stated above, rigorous economic analyses of e.g. the competition between OS software and Microsoft are still hard to find. However, there is a discussion of this topic which has been made public in a less formal setting and which has received considerable attention within the open source community. This discussion is contained in the memos which have become known as the "Halloween documents". These memos have been written by a Microsoft representative, and they have originally been meant for the internal use of Microsoft.[10]

[8] Ibid., p. 3.
[9] Schmidt – Schnitzer (2002), p. 8.
[10] In the response of Microsoft to the discussion that the so-called "Halloween documents" have created a Microsoft representative (enterprise marketing group manager Ed Muth) has subsequently stated that the documents in question are genuine in the sense of being "technical analyses written by a staff engineer that represent the thoughts of one individual at one point in time". However, in the same response it is also emphasized that they "do not represent an official Microsoft position or road map" (quotation taken from an archived version of a page which has been available on the Microsoft internet portal, from the address http://web.archive.org/web/20010211142733/http://www.microsoft.com/ntserver/nts/news/mwarv/linuxresp.asp accessed on August 14, 2002).

In the so-called "Halloween Documents", which the non-profit corporation "Open Source Initiative" has subsequently made available on the Internet, it is stated that OS software constitutes a threat to the current dominant position of Microsoft. According to what has become called "Halloween Document I", "OSS poses a direct, short-term revenue and platform threat to Microsoft -- particularly in server space. Additionally, the intrinsic parallelism and free idea exchange in OSS has benefits that are not replicable with our current licensing model and therefore present a long term developer mindshare threat."[11] The author of the so-called "Halloween Document I" also admits the efficiency of parallel debugging and some other principles which Eric S. Raymond has formulated and on which OS development is based. On the other hand, the document also contains views which sharply contradict those of Raymond: for example, it is claimed in the document that the OS development model is efficient only in situations in which the OS project is able to "chase taillights", in other words when the product which the OS project is trying to produce has similar functions with other already existing products which are of a higher quality. In addition, the document also on the one hand points out that the "process around" Linux "lends credibility and an air of future-safeness to customer Linux investments" but, on the other hand, emphasizes the inability of OS projects to make legally binding commitments.[12]

The relationship between commercial software producers and OS software producers is discussed in a more systematic manner in Lerner – Tirole (2002). As Lerner and Tirole point out, the features of OS software projects on which their success is based are such that the commercial software producers have no easy way of reproducing them.[13] As we mentioned above in subsection 3.2, one of such features is the so-called "*alumni effect*": OS programming environments are often used in universities and schools for learning purposes, and for this reason the persons who start to contribute to an OS project are often more familiar with its programming environment than the programmers who are hired by a commercial company are with the environment within which they start working. Secondly, commercial software companies cannot make an efficient use of users' incentives to customize the software and to fix bugs in it as OS projects do. And thirdly, the *signaling incentives* which motivate programmers are not as strong in the field of commercial software production as they are in OS projects.[14] Although also commercial companies can e.g. publish a list of the employees who have contributed to some particular project, outsiders will not able to find out which tasks each employee has completed, and this will make the claims concerning the importance of her or his work unverifiable.[15] In addition, a commercial company might be unwilling to increase the "visibility" of its key employees because this would increase the danger that they would be hired by competitors.[16]

Nevertheless, a commercial software company might wish to emulate some features of OS software production. One of the ways of doing this is to promote *code sharing* within the company and among customers. For example, also Microsoft provides a part of its customers

[11] Halloween Document I.
[12] Ibid..
[13] Lerner – Tirole (2002), p. 223.
[14] Ibid.
[15] Ibid., pp. 223-224. However, as we stated in subsection 3.2 above, this view concerning signaling incentives could be criticized by pointing out that, on the one hand, also many commercial companies try to measure the quality of the work of their employees and that, on the other hand, the visibility of the contributions of OS software programmers has been reduced by the introduction of more liberal OS licenses.
[16] Ibid., p. 224.

with the source code of Windows.[17] A commercial company might, of course, try to benefit from the existing open source software also more directly. As Lerner and Tirole point out, there are several ways in which commercial companies might do this.

The most obvious way in which a commercial company might utilize OS software financially is to *live symbiotically with an OS project*, i.e. to provide services and products that are complementary to an OS software product and which the OS project does not itself produce efficiently.[18] For example, the company *Red Hat* follows this strategy by selling its own versions of OS products like Linux operating system and the Apache web server. Although these OS products are available on the Internet for free, Red Hat motivates its customers to buy them from it by providing documentation and support for them.[19]

Another strategy which a commercial company may employ is to make its proprietary source code available under an open source license.[20] The most famous example of this procedure has been provided by the Netscape Communications Corporation. Netscape has made "Mozilla", the body of the source code of its web browser *Netscape Communicator*, freely available. The background of the decision to do this and its consequences are discussed in some detail in Hamerly – Paquin – Walton (1999). While discussing the background of this decision, Hamerly, Paquin, and Walton mention the fact that Netscape had already earlier chosen to make its web browser, Netscape Navigator, available for free on the Internet and state that many Netscape employees had earlier experience of working with Open Source.[21] One of the difficulties with which the publication of the source code was confronted was caused by the fact that contained "third-party modules", i.e. pieces of code which not owned by Netscape, and it was necessary to find an agreement with the owners of these "modules" or replace them with new ones.[22]

The choice of an appropriate license was associated with problems of another kind. Among the existing OS licenses *the BSD license* seemed too permissive, because it seemed to allow the creation of modified versions of the code which would not be made public to the OS community, whereas the *GPL license* was viewed as unsuitable because it is *viral* in the sense that also other pieces of code which utilize source code to which it applies must be licensed under the GPL license. Netscape resolved this problem by creating two licenses of its own, Netscape Public License and Mozilla Public License which differ from each other in so far that the former grants special rights to Netscape and the latter does not.[23] It was also necessary to decide how an open source project ran by a commercial company would be led. Netscape dealt with this problem by creating an organization called "Mozilla.org" for managing the OS project it had created. As Harmerly, Paquin, and Walton point out, it was quite essential for the credibility of this project that Mozilla.org was kept separate from Netscape Client Product Development Group, which served the commercial interests of Netscape.[24]

[17] See e.g. Mundie (2002), pp. 8-9, in which the views of Microsoft concerning sharing source code are presented in some detail. Cf. Lerner – Tirole (2002), p. 224.
[18] Ibid., pp. 224-5.
[19] Cf. the internet portal of Red Hat (http://www.redhat.com).
[20] Lerner – Tirole (2002), pp. 225-7.
[21] Hamerly et al. (1999), pp. 197-198.
[22] Ibid., pp. 199. On a more specific level, Hamerly et al. mention the problems which were caused by the fact that the Java language had been used in Mozilla. Java was a proprietary language, and it could not be used in the open source version. (Ibid.)
[23] Ibid., pp. 200-202.
[24] Ibid., pp. 203-204.

On the other hand, *Brian Behlendorf*, a co-founder of the Apache Group, has discussed in more general terms the question what kind of software items a commercial company could reasonably turn into open source. He states that there have until now been more open source products in the field of "infrastructural" software – i.e. software implementing "frameworks or platforms" – than in the field of end-user applications. Accordingly, Behlendorf suggests is that a commercial company which turns a part of the proprietary software which it owns into open source software should do this to its *infrastructural software* rather than to end-user applications,[25] which are exemplified by the Netscape Communicator which we considered above.

A third way in which a commercial company could participate in OS development is to act as an intermediary between OS programmers and commercial companies. Lerner and Tirole mention *CollabNet* as an example of a company with a business idea of this kind.[26] According to its Internet portal, Collabnet offers for commercial open source projects a platform which includes discussion forums, a web-based project environment, a service of notifying the appropriate experts on the various issues that arise during the project etc.[27]

Lerner and Tirole also discuss the question whether commercial companies could have an incentive for contributing to open source projects. Companies like Red Hat, whose sales depend directly on the quality of the available open source software, are the most obvious examples of companies which might have such an incentive. However, there is a less direct sense in which all the software companies which compete with Microsoft with products that are complementary to operating systems might be claimed to benefit from the popularity of Linux. This is because its popularity decreases the total amount of money which is spent on operating systems, and this can be expected to increase the amount of money that customers are willing to spend on items – like on computer hardware and on software products of other kinds – which are complementary to them. The software companies which compete with Microsoft might even be motivated to hire employees for making contributions to OS projects, like e.g. for making the Linux operating system compatible with their products. However, the willingness of commercial companies to make such contributions is, of course, diminished by free-rider problems.[28]

Klaus Schmidt and *Monika Schnitzer* have addressed in Schmidt – Schnitzer (2002) an important topic within the economic analysis of the competition between commercial software producers and open source projects. This is the question whether OS projects should be given *government subsidies*. Schmidt and Schnitzer answer this question negatively. Such subsidies would, obviously, be a part of the research and development expenditures of the government, and it is the view of Schmidt and Schnitzer that, in general, governments should restrict themselves to subsidizing basic research rather than applied research and development.[29] They motivate this view by three arguments. According to their first argument a profit maximizing firm has a strong incentive to develop products that are adapted to the needs of its customers, but a government sponsored research lab has less incentive to do so. Secondly, Schmidt and Schnitzer point out that public subsidies invite rent seeking activities: projects will try to get funds for themselves by lobbying rather than by competing with each other on the market. The

[25] Behlendorf (1999), pp. 158-160.
[26] Lerner – Tirole (2002), p. 227.
[27] See the internet portal of Collabnet (http://www.collab.net, addressed on August 13, 2000).
[28] Lerner – Tirole (2002), pp. 224-225.
[29] Schmidt – Schnitzer (2002), p. 22.

third argument is concerned with markets in which there are strong network externalities, as there are in the software market. In such markets even small subsidies can have very dramatic effects: they can e.g. make the market "tip" in the sense that the government sponsored product receives an almost monopolistic position. [30]

As a fourth point, Schmidt and Schnitzer state that if the government nevertheless chooses to subsidize software development, it should make sure that the resulting publicly sponsored software is put into the public domain or protected by a liberal open source license.[31] They also argue against the policy of prohibiting government agencies from using proprietary software if an open source alternative exists: they point out that in some cases open source software might not be the most cost effective solution, that also this policy might make the market "tip" when there are strong network externalities,[32] and that if the decision to force government agencies to use OS software was motivated politically – rather than by comparisons of software quality – this decision might actually reduce rather than increase competition in the software market.[33]

5.3 OS Software and Software Quality

We shall conclude this section by discussing shortly the effects of the emergence of OS software on software quality. The relevant empirical evidence, which is provided by e.g. quality comparisons between Linux and Microsoft Windows, is often controversial.[34] One of the factors which make such quality comparisons difficult is the fact that seemingly comparable open source and proprietary software products might, as a matter of fact, have been meant to serve different customer groups. For example, according to Lerner and Tirole "many open source advocates" argue that OS software "tends to be geared to the more sophisticated users", whereas Microsoft has to take also its more ignorant customers into account.[35]

Steven Weber addresses one of the aspects of quality comparisons between open source and proprietary software in Weber (2000) by pointing out that if Linux manages to overturn the equilibrium in the software market, in which Microsoft currently has almost a monopoly as a producer of operating systems, one might view it as a way of overcoming a suboptimal situation which persists because of *path dependence*.[36] According to him, "other operating systems (primarily Microsoft products) are sub-optimal performers but are locked in by increasing returns and high costs of switching".[37] In other words, e.g. the current versions of Microsoft Windows are according to Weber not optimal operating systems because it has been necessary for Microsoft to make the newer versions of Windows compatible with its earlier versions and even with some features of MS-DOS.

[30] Ibid., p. 23.
[31] Ibid., p. 24.
[32] Ibid., p. 24-25.
[33] Ibid., p. 29.
[34] See e.g. Moody (2001), pp. 274ff.
[35] Lerner – Tirole (2002), p. 203.
[36] Weber (2000), pp. 36-37.
[37] Ibid., p. 37.

Weber states that if Linux overturns the equilibrium in the microcomputer operating system market, "the open source community will have demonstrated one way to successfully invest in a jump to a higher performance path".[38] It would, obviously, be an essential feature of this way of investing that the contributors of Linux have no such obligations towards the users of Linux as commercial companies have towards their customers, like making Linux compatible with some earlier operating systems. Since contributing to Linux is voluntary, according to Weber "[t]raditional economic analysis will suggest that this was the outcome of subsidization",[39] but in the case of Linux the subsidization would have been of a rather unusual kind. Weber also points out that open source operating systems might be confronted with similar path dependency problems later on and that it is currently still difficult to evaluate how well OS communities are able to cope with them. As Weber states, it might actually be *more difficult*, rather than easier, for a non-hierarchical community to e.g. make a decision to move to fundamentally new operating system architecture than making such a decision is when there is a "boss" who has the authority for ordering that the change must be made.[40]

An important aspect of the quality comparisons between open source and proprietary software is constituted by the comparisons of their *reliability* and *security*. In Section 4 above we already presented some reasons which have been claimed to make OS software particularly reliable. According to an aphorism which is due to Eric S. Raymond, "Given enough eyeballs, all bugs are shallow".[41] In other words, when a program has a sufficient number of beta-testers who use it and who search for bugs in the source code when it behaves in an unsatisfactory way, all the problems with the source code will be characterized quickly and also the solutions to them will be found quickly.[42] On the other hand, *Craig Mundie*, a representative of Microsoft, has in the recent paper Mundie (2002) compared the security of commercial and open source software and claimed that "a balanced comparison of the number and severity of security vulnerabilities in commercial and open source software would show rough parity between the two types of software".[43] He states that it is "Microsoft's position" that the "OSS model" does not "*inherently* lead to more secure software"[44] and that, for example, there have been repeated "discoveries of security vulnerabilities in shared open source components such as Sendmail, WU-FTPD and BIND".[45]

Mundie also objects to the view that the security of OS software would be superior to that of commercial software because of "the pride that open source programmers feel in writing good code" and because of "the lack of commercial pressure to 'do it fast' rather than 'do it right'". He claims that, because of the importance that security has for many customers, "commercial software companies are under much greater pressure to 'do it right'".[46] Mundie claims that the "broad availability of source code becomes a double-edged sword" in the sense that it "provides a well-resourced and motivated party the means to conceive increasingly sophisticated attacks".[47] In other words, the public availability of source code might be useful

[38] Ibid.
[39] Ibid.
[40] Ibid.
[41] Raymond (2001), p. 30.
[42] Cf. Ibid.
[43] Mundie (2002), p. 4.
[44] Ibid., p. 3.
[45] Ibid., p. 5.
[46] Ibid., p. 4.
[47] Ibid., p. 6.

also for e.g. unauthorized persons who are making an attempt to break into a computer system, because such persons might make a detailed study of the source code of the software which is used in the system and find in it bugs which enable them to break into it.

Ross Anderson has presented a model for the reliability growth of open source and proprietary software. Also according to Anderson "open and closed systems", which are exemplified by open source software and proprietary software, respectively, will in general "exhibit similar growth in reliability and in security assistance".[48] This is result seems quite obvious in the context of Anderson's model, in which the time it takes for a bug to appear is viewed as random variable with an exponential distribution. In this case it is clear that both open source projects and the producers of proprietary software will fix the more often appearing bugs themselves and that they have to delegate the finding of the less usually appearing bugs to the users of the software.[49,50]

[48] Anderson (2002), p. 4.
[49] Cf. ibid., pp. 2-4.
[50] *Hal R. Varian* has presented an interesting perspective on related issues in Varian (2002), in which system reliability is viewed as a *public good*. He discusses the production of this public good by a number of individuals and distinguishes between three prototype cases: in the first of these system reliability depends on the sum of the efforts of the individuals, in the second one it depends on the smallest effort made by them, and in the third one it depends on the largest effort made by them (p. 1). Varian states that his "motivating example" is "computer system reliability and security", which depends on the efforts expended by "teams of programmers and system administrators" (p. 2). It seems that among his three "paradigm cases" the reliability of open source software resembles most closely the case in which the *sum* of the efforts is essential. For example, it is not of crucial importance for open source projects that some individual contributions to them might be of a very low quality, since when this happens, a part of the efforts of the other contributors will expended on improving them.

6. Concluding Remarks

Above we surveyed some economic perspectives on OS software. After a short introduction to this field of study, we proceeded to a discussion of the microeconomic analyses of the motives of the individuals who participate in OS projects, the analyses of the structure of OS software projects, and the currently still rather rudimentary macroeconomic analyses of the competition between OS software and ordinary proprietary software. As we have seen, OS software is in many ways a puzzling phenomenon for the economist, and its emergence raises many interesting questions which we have not covered above. Some of these have to do with the significance that OS software has specifically for the *developing countries*.

The fact that open source and open code software is freely available, and it can accordingly be used by persons who cannot afford to use proprietary software, is the most obvious of the reasons which make OS and OC software important specifically for the developing countries.[1] Somewhat less obviously, the increasing popularity of the kind of *modular structure* which OS software projects have introduced for their products might change the whole field of software production dramatically, and such changes could be particularly important for the developing countries. As we stated in the beginning of Section 4 above, software which has such structure consists of relatively small and self-contained modules which communicate with each other with clearly specified communication interfaces. The increasing modularity of OS software might lead to a situation in which e.g. government agencies and small companies could with a relatively small amount of work build from the available OS software components software products which are specifically customized for their purposes.

In his discussion of the significance of OS software for the developing countries Steven Weber has considered even more radical scenarios. Weber distinguishes between two possible future developments in the early 21st century which differ with respect to the role which software production has in the economy. In the first of the two cases software production is "'merely' a new leading sector of the economy", and in the second one it turns out to be "more profoundly a transformative tool that sweeps across and revolutionizes economic activity across a very wide range of sectors".[2] In the former case the significance of open source software will for the most part be restricted to computing and information processing, but in the latter case "it is reasonable to expect new possibilities for organization to arise across the economy".[3]

On a more specific level, Weber claims that "open source software is likely to be a powerful instrument of bootstrapping",[4] since in the field of open source software development contributors from the developing countries are in an equal position with those from the developed ones. In particular, the contributors from the developing countries are able to decide themselves what kind of OS software they produce and use, which contrasts sharply with the

[1] The importance of the free availability of OS software has recently been illustrated in a interesting way in a report of the National Advisory Council on Innovation of South Africa. This report contains several example scenarios which illustrate the ways in which open source software can be used by South Africans who cannot afford proprietary software e.g. for accounting or teaching purposes (National Advisory Council on Innovation, 2002, p. 5-6), as well as a discussion of a project which has translated OS software into African languages (ibid., p 8; cf. the internet portal of the organization Translate, http://www.translate.org.za.)

[2] Weber (2000), p. 39.

[3] Ibid.

[4] Ibid.

"post World War II era of development economics", during which "developed country governments and international development institutions were making decisions about what was 'appropriate technology' to transfer to developing countries".[5] Weber illustrates his views also by considering an imagined situation in which "everyone had access to as many steam engines as they wanted, all at the same time, and for nearly no cost". This would not have removed the unevenness of development during the industrial era, but the development "very well might have been *less drastically uneven* than it is today".[6]

As we pointed out in Subsection 3.2 above, the current situation in the field of OS production does not support Weber's views, since currently OS software is for the most part produced in the developed parts of the world. An economist would expect that programmers would be willing to contribute to OS projects when the opportunity costs of contributing to them are low which, one might think, would be the case specifically in the developing countries. Accordingly, it is interesting to ask why OS software is currently for the most part produced in the developed parts of the world. In Section 1 we pointed out that a possible answer to this question would be to claim that the "social and political structures" which enable OS production would not be present in the developing countries to the same extent as in the developed ones. Above we described some aspects of these structures, as we e.g. in subsection 4.3 emphasized the role of competent *managers* for the success of an OS project and subsection 3.2 discussed the significance of an OS *community*, which often consists of persons who identify themselves with the so-called "hacker culture". A possible explanation for the lack of OS production in the developing countries would be given by the claim that such communities would not exist in them to the same extent as they do in the developed countries. This supposition leads to another interesting question, which we considered shortly in subsection 5.2, i.e. the question whether the developing countries should subsidize OS projects which are specifically suited for their needs. However, a more detailed analysis of these issues lies beyond the scope of this paper.

[5] Ibid.

[6] Ibid., pp. 39-40.

References

Aghion, Philippe – Howitt, Peter (1992): "A Model of Growth through Creative Destruction". *Econometrica* **60**, 323-351.

Anderson, Ross (2002): "Security in Open versus Closed Systems – The Dance of Boltzmann, Coase and Moore". A paper presented at the conference "Open Source Software: Economics, Law and Policy", Toulouse, France, June 20-21, 2002 (http://www.idei.asso.fr/ossconference.html).

Behlendorf, Brian (1999): "Open Source as a Business Strategy". In DiBona – Ockman – Stone (eds.) (1999), pp. 149-170.

Bergstrom, Theodore – Blume, Lawrence – Varian, Hal (1986): "On the Private Provision of Public Goods", *Journal of Public Economics* **29**, 25-49.

Bliss, Cristopher – Nalebuff, Barry (1984): "Dragon-Slaying and Ballroom Dancing: the Private Supply of a Public Good", *Journal of Public Economics* **25**, 1-12.

Brooks, Frederick P. (1975): *The Mythical Man-Month. Essays on Software Engineering*. Addison Wesley, Reading, Mass.

Dafermos, George N. (2001): "Management & Virtual Decentralised Networks: the Linux Project". Paper available on the Internet portal of the *Open Source Research Community* at the address http://opensource.mit.edu/papers/dafermoslinux.pdf. Accessed on August 7, 2002.

DiBona, Chris – Ockman, Sam – Stone, Mark (eds.) (1999): *Open Sources. Voices from the Open Source Revolution.* O'Reilly, Beijing.

DiMaggio, Paul J. – Powell, Walter W. (1983): "The Iron Cage Revisited: Institutional Isomorphism and Collective Rationality in Organizational Fields". *American Sociological Review* **48**, 147-160.

Economides, N. (1996): "The Economics of networks", *International Journal of Industrial Organization* **14**, 673-699.

Edwards, Kasper (2001): "Epistemic Communities, Situated Learning, and Open Source Software Development". Paper available on the Internet portal of the *Open Source Research Community* at the address http://opensource.mit.edu/papers/kasperedwards-ec.pdf. Accessed on August 5, 2002.

Franck, Egon – Jungwirth, Carola (2002): "Reconciling investors and donators – The governance structure of open source". *Lehrstuhl für Unternehmensführung und –politik, Universität Zürich, Working Paper Series*, Working Paper No. 8.

Ghosh, Rishab Aiyer (2002): “Clustering and Dependencies in Free/Open Source Software Development: Methodology and Preliminary Analysis”. A paper presented at the conference “Open Source Software: Economics, Law and Policy”, Toulouse, France, June 20-21, 2002 (http://www.idei.asso.fr/ossconference.html).

Gravelle, Hugh – Rees, Ray (1994): *Microeconomics. Second Edition.* (First published 1981, second edition 1982). Longman, London.

Grossman, Gene M. – Helpman, Elhanan (1991): *Innovation and Growth in the Global Economy.* The MIT Press, Cambridge, Mass.

*The Halloween Memos***.** The so-called “Halloween Memos” are quoted in the form they appear on the Internet portal of the *Open Source Initiative*, at the address http://www.opensource.org/halloween. (Accessed on August 9, 2002.)

Hamerly, Jim – Paquin, Tom – Walton, Susan (1999): “Freeing the Source: The Story of Mozilla”, in DiBona et al. (1999), pp. 197-206.

Hardin, Russell (1982): *Collective Action.* The John Hopkins University Press, Baltimore.

Harhoff, Dietmar – Henkel, Joachim – von Hippel, Eric (2000): “Profiting from voluntary information spillovers: How users benefit by freely revealing their innovations”, *MIT Sloan School of Management Working Paper* 4125. (Loaded on July 18, 2002, from the address http://opensource.mit.edu/papers/evhippel-voluntaryinfospillover.pdf)

Von Hippel, Eric (1988): *The Sources of Innovation*. Oxford University Press, New York.

Von Hippel, Eric (1994): “ ‘Sticky Information’ and the Locus of Problem Solving: Implications for Innovation.”, *Management Science* 40/4, 429-439.

Von Hippel, Eric (2001): “Innovation by User Communities: Learning from Open-Source Software”. *Sloan Management Review* **42/4**, 82-86.

Von Hippel, Eric (2002): “Open source software projects as user innovation networks”. A paper presented at the conference “Open Source Software: Economics, Law and Policy”, Toulouse, France, June 20-21, 2002 (http://www.idei.asso.fr/ossconference.html).

Von Hippel, Eric – von Krogh, Georg (2002): “Exploring the Open Source Software Phenomenon: Issues for Organization Science”. Paper available on the Internet portal of the *Open Source Research Community* at the address http://opensource.mit.edu/papers/hippelkrogh.pdf. Accessed on August 8, 2002.

Holmström, Bengt (1999): “Managerial Incentive Problems: A Dynamic Perspective”. *Review of Economic Studies* **66**, 169-182.

Johnson, Justin Pappas (1999): "Essays in Microeconomic Theory". Thesis (Ph.D.) at the Massachusetts Institute of Technology, Department of Economics, Cambridge, Mass.

Kelty, Christopher M. (2001): "Free Software / Free Science", *First Monday* **6,** number **12**.

Kollock, Peter (1999): "The economies of online cooperation: gifts and public goods in cyberspace", in Smith – Kollock (eds.) (1999), pp. 220-239.

Von Krogh, Georg (1998): "Care in Knowledge Creation". *California Management Review* **40/3**, 133-153.

Kuan, Jennifer (2002): "Open source Software as Lead User's Make of Buy Decision: A Study of Open and Closed Source Quality". A paper presented at the conference "Open Source Software: Economics, Law and Policy", Toulouse, France, June 20-21, 2002 (http://www.idei.asso.fr/ossconference.html).

Kuwabara, Ko (2000): "Linux: a Bazaar at the Edge of Chaos", *First Monday* **5**, number **3**.

Lakhani – von Hippel (2002): "How Open Source software works: 'Free' user-to-user assistance". Paper available on the Internet portal of the *Open Source Research Community* at the address http://opensource.mit.edu/papers/lakhanivonhippelusersupport.pdf. Accessed on August 27, 2002.

Lancashire, David (2001): "Code, Culture and Cash: The Fading Altruism of Open Source Development", *First Monday* **6**, number **12**.

Lerner, Josh – Tirole, Jean (2002): "Some Simple Economics of Open Source". *The Journal of Industrial Economics*, **50**, 197-234.

Levy, Steven (1984): *Hackers: Heroes of the Computer Revolution.* Penguin Books, New York.

Mintzberg, Henry (1979): *The Structuring of Organizations. A Synthesis of the Research.* Prentice-Hall, Inc., Englewood Cliffs.

Mockus, Audris – Fielding, Roy T. – Herbsleb, James (2000): "A Case Study of Open Source Software Development: The Apache Server", in *Proceedings of the 2000 International Conference on Software Engineering*, Los Alamitos, IEEE Computer Society, 263-272.

Moody, Glyn (2001): *Rebel Code.* Perseus Publishing, Cambridge, Mass.

Moon, Jae Yun – Sproull, Lee (2000): "Essence of Distributed Work: The Case of the Linux Kernel", *First Monday* **5**, number **11**.

Mundie, Craig (2002): "Security: Source Access and the Software Ecosystem". A paper presented at the conference "Open Source Software: Economics, Law and Policy", Toulouse, France, June 20-21, 2002 (http://www.idei.asso.fr/ossconference.html).

National Advisory Council on Innovation (2002): "Open Standards and Open Source Software in South Africa". A discussion document of the *National Advisory Council on Innovation*, South Africa, January 2002. Available at the Internet address http://www.naci.org.za/docs/opensource.doc (accessed on August 16, 2002).

Palfrey, Thomas R. – Rosenthal, Howard (1984): "Participation and the Provision of Discrete Public Goods: a Strategic Analysis". *Journal of Public Economics* **24**, 171-193.

Pressman, Roger S. (2000): *Software Engineering. A Practitioner's Approach*. (Adapted by Darrel Ince.) McGrawHill, London.

Raymond, Eric S. (2001): *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Revised Edition. O'Reilly, Beijing.

Romer, Paul M. (1990): "Endogenous Technological Change". *Journal of Political Economy* **98**, S71-S102.

Rosenberg, Nathan (1976): *Perspectives on Technology*. Cambridge University Press, Cambridge.

Saint-Paul, Gilles (2002): "Growth Effects of Non-Proprietary Innovation". A paper presented at the conference "Open Source Software: Economics, Law and Policy", Toulouse, France, June 20-21, 2002 (http://www.idei.asso.fr/ossconference.html).

Schmidt, Klaus M. – Schnitzer, Monika (2002): "Public Subsidies for Open Source? Some Economic Policy Issues of the Software Market". A paper presented at the conference "Open Source Software: Economics, Law and Policy", Toulouse, France, June 20-21, 2002 (http://www.idei.asso.fr/ossconference.html).

Shapiro, C. - Varian, H. R. (1999): *Information Rules. A Strategic Guide to the Network Economy*. Harvard Business School Press, Boston.

Smith, Marc A. – Kollock, Peter (*eds.*) (1999): *Communities in Cyberspace*. Routledge, London.

Stallman, Richard (1999): "The GNU Operating System and the Free Software Movement", in DiBona *et al.* (eds.) (1999), pp. 53-70.

Torvalds, Linus (1999), "The Linux Edge", in DiBona et al. (eds.) (1999), pp. 101-111.

Weber, Steven (2000): “The Political Economy of Open Source Software”. *Berkeley Roundtable on the International Economy Working Papers* **140**, University of California, Berkeley. Quoted with the permission of the author.

Van Wendel de Joode, Ruben – Kemp, Jeroen (2002): “The Strategy Finding Task within Collaborative Networks, based on an exemplary Case of the Linux Community”. A paper presented at the 3rd IFIP Working Conference on Infrastructures for Virtual Enterprises, Sesimbra, Portugal.

Zhang, Wei – Storck, John (2001): “Peripheral Member in Online Communities”. Paper available on the Internet portal of the *Open Source Research Community* at the address http://opensource.mit.edu/papers/zhang.pdf. Accessed on August 7, 2002.