CPB Discussion Paper

No 13

September 2002

Open vs. closed: Some consequences of the open source movement for software markets.

Ewa Mendys - Kamphorst

CPB Netherlands Bureau for Economic Policy Analysis
Van Stolkweg 14
P.O. Box 80510
2508 GM The Hague, the Netherlands

Telephone	+31 70 338 33 80
Telefax	+31 70 338 33 50
Internet	www.cpb.nl

ISBN 90-5833-112-1

Contents

Abstract.

Open source movement has received more and more attention in the recent years. However, there are not many studies which analyse its economic impact in software markets. This paper gives a description of the open source phenomenon and attempts to analyse its impact on competition. The focus is on consequences for entry. The comparison of open source projects with proprietary firms turns out to be difficult because of a different nature of incentives which motivate both types of developers. It is therefore difficult to describe the general impact of open source. Nonetheless, one can try to classify open source programs according to their expected effects. The analysis is illustrated with examples, with a particular emphasis on the competition between Microsoft Windows and GNU/Linus operating system.

1 Introduction

The open source "movement" and open source projects have received more and more attention in the recent past. For instance, The *Economist* mentions one of the best known open source programs, the Linux operating system, in 3 articles in 1998, 14 articles in 1999, 27 in 2000, and 11 articles between 1.01 and 30.04 2001. This rise in interest is accompanied by an increasing market share of open source software. On the other hand, the initial (1998 - 1999) enthusiasm for the open source business model cooled down after significant losses suffered in 2000 by many companies that in one way or another based their business strategies on open source.

The opinions on the welfare effects of open source movement are divided. According to some, open source means more freedom to consumers, products scrutinised by hundreds of programmers and therefore with fewer errors and the end of monopoly power of a few large software companies, notably Microsoft. Others point out the dangers of open source: insufficient standardisation and lowering the incentives to innovate by decreasing the industry's profits (see e.g. the speech by Craig Mundie, Microsoft Senior Vice President at The New York University Stern School of Business, May 3, 2001).

This paper aims at introducing the reader to the phenomenon of open source, and at identifying some of the effects that open source is likely to have in software markets. The focus of the economic analysis is on the impact on competition (with prices and entry possibilities as its indicators), the investment in quality, and the choice of the level of differentiation and compatibility, both between programs made by different producers and between versions of the same program.

The term "open source" was introduced in 1997 by the Open Source Definition (see the history section) as an alternative to the formerly used "free software". The word "free" was used there to describe the expanded rights of users, which included not only the right to use the program (which the users of commercial programs also have), but also to modify and redistribute it, possibly for a fee. The term "free software" is still in use (as in Free Software Foundation), but since it had been associated with an anti - business approach, some proponents of open source (e.g. Eric Raymond[1]), came up with "open source" as a part of a campaign designed to overcome the distrust of the corporate world.
Open source software (OSS) can be defined as software that is distributed not only as binary code ready for running, but in addition together with its source code, that is the actual program written by a developer in a particular programming language. Most commonly, the source code is available for downloading via the Internet, although the more popular programs are also often distributed on CDs. For instance, Linux is distributed by companies like Red Hat, VA Linux,

[1] Creator of the open source e-mail program “fetchmail” and author of essays about open source, among which the most famous one, “The Cathedral and the Bazaar” (first published in 1997).

SuSE etc. The programs are freely available and users are allowed to copy, change and redistribute the program, including reselling. There are, however, certain restrictions on the users of programs distributed under one of the open source licenses. Most importantly, no user is allowed to appropriate the code and close it from other users and, depending on the license, s/he may or may not be obliged to make own contributions to the program available to others.

The availability of the code distinguishes open source from *freeware or shareware*, and licensing makes it different from *public domain* software. The former is distributed for free, but the code remains the property of its developer, and the latter is not copyrighted and places no restrictions on users.

Open source software is developed co-operatively by numerous programmers, who contribute voluntarily to the development. That means that no financial compensation is offered to the authors of enhancements. The projects have their own development sites on the Internet, and the contributions are sent in via the net. The contributions are usually examined by a smaller group of project leaders (which can consists of only one person), whose typical role is to assess submissions and decide about their inclusion in the program. The leader(s) are often the one(s) who create the initial program - which can be further developed and improved - and who provide the general vision about the eventual properties of the program.

The rest of the paper is organised as follows. Section 2 gives a short overview of the open source history, and Section 3 contains some stylised facts about open source. Section 4 focuses on competition and the use of investment in quality, differentiation and compatibility as instruments of entry deterrence and accommodation, while Section 5 discusses the problem of standardisation between versions of the same program. When possible, the analysis is illustrated with examples. Section 6 concludes and discusses the social welfare implications. The Appendix contains more detailed information on selected open source programs.

2 History[2]

From the 1960s to early 1980s, at the time when computer programming was still in its youth, most operating systems were developed in the way embraced now by the open source movement. The main centres of development were universities and autonomous research labs, such as Artificial Intelligence Laboratory at MIT, Stanford University Artificial Intelligence Lab, Carnegie Mellon University and Xerox Palo Alto Research Centre. Operating systems were developed for own use, separately for different types of hardware and highly customised, and cheap personal computers were not available. Hence, a mass market for standardised software did not exist. In the absence of commercial motivation, the developing institutions had incentives to share their work with others, especially with those who used the same hardware (many of the above mentioned labs used PDP machines of Digital Equipment Corporation, with DEC software on it). The sharing was facilitated by the creation of first long-distance networks, like ARPAnet in 1969, the first transcontinental, high-speed network, originated at the Defence Department, soon connecting many universities and research labs.

The year 1969 saw the creation of two programs that significantly changed the way in which programming was done. Two employees of Bell Labs of AT&T invented Unix operating system (Ken Thompson) and C programming language (Dennis Ritchie). Because Unix could be entirely written in C, and not only, as in the case of previous operating system, in the hardware-specific assembler, it could be run on different machines. Moreover, both Unix and C had a fairly simple and flexible structure. That led to their fast diffusion, and by 1980 they had spread to many universities and research sites, o worked further on their development and shared the code among each other. The creation of a new Unix-network, Usenet, in 1980, led to the further rise in the developers' community. At the beginning of the 80s Unix became the main operating system.

In the meantime, the first microcomputers had been introduced, which made computers available for a broader public. The growth of the market together with the broad adoption of the same operating system created, for the first time, a significant demand for a standardised Unix. In an attempt to profit from these trends, AT&T began to enforce its property rights to Unix, which then became a commercial product for the first time[3].

This led to protests from a part of programmers' community, who had helped to develop the program. As a reaction, and to prevent similar situations in the future, some programmers united, under the leadership of Richard Stallman from MIT Artificial Intelligence Laboratory, in the Free Software Foundation (FSF) that started in 1983. The goal of the FSF was to develop and

[2] See also Raymond (2000a), and Lerner and Tirole (2000).

[3] Which it remains until today.

distribute free software, including its source code, and in particular the creation of an open source operating system, called GNU ("GNU's not Unix").[4]

In order to prevent "hijacking" of their work and its commercialisation in the way it happened with Unix, the Foundation invented a General Public License (GPL, "copylefting"), with which all the users of their software have to comply. Software licensed under GPL can be used, changed and redistributed freely, but its source code has to be made available to everyone and no restrictions (other than imposed described by the GPL itself) can be placed on its use. Moreover, all derived work, including bug fixing, extensions and other enhancements, has to be made public, as well as the code of all programs distributed together with a copylefted program.

Open source activity speeded up at the beginning of the 90s as a consequence of improved and more widely available communication via the Internet. New projects emerged, including Linux. Alternative licensing arrangements were developed, less restrictive than GPL, e.g. Debian Social Contract under which developers are allowed to bundle free and proprietary code together.

The Debian Social Contract was the basis for the Open Source Definition (OSD) license, developed in 1997 by among others Eric Raymond (the chief "ideologist" of the open source movement), Tim O'Reilly (the publisher of manuals and books about open source programs). Their goal was to popularise open source software, and one of the steps towards it was overcoming the anti-business image of free software and the Free Software Foundation. To make open source more attractive for businesses the OSD license allowed bundling with proprietary software and abandoned the obligation to make all changes public. The creation of the OSD was coupled with an intensive publicity, conducted for a large part by Eric Raymond, whose essay "The Cathedral and the Bazaar" (analysing the history of the development of an open source e-mail program, e-macs) and its sequels, "Homesteading the Noosphere" and "The Magic Cauldron" became open source manifestos.

Since 1997, open source software has aroused more and more interest in the IT world. In 1998, Netscape opened the source of their Internet browser, Netscape. Projects proliferated, and more and more businesses adopted or consider adopting open source software for their needs. A limited number of open source programs gained dominant market shares (e.g. Apache obtained ca 60% of the web servers market, and Sendmail ca 80% of the market for e-mail servers), and a number of businesses were built around distributing versions of Linux.

There are not many aggregate data on the diffusion and successfulness of open source projects. There are usually no sales figures as for commercial programs, and even for programs that are distributed on CDs, like Linux, the figures are likely to underestimate their actual use, since they can be freely copied or downloaded. For many other programs only download figures

[4] It did not succeed in creating an integraded operating system, but many of the partial solutions have been combined with the Lenux kernel (core) to form the GNU/Linux.

are available, which convey limited information about their actual popularity because not all downloaded copies are actually used, and on the other hand users can make copies from one another.

Some data can be found on the development sites of the open source programs. For instance, the SourceForge service (sourceforge.net) run by Open Source Development Network hosts about 22,500 thousand projects and has almost 200,000 registered users (22,676 and 199,675 respectively on 28.06.2001), of which the largest have been downloaded almost 600,000 times, and 70 of the programs have been downloaded by more than 20,000 users (data from 28.06.2001). For some programs survey data are available, for instance for Apache, which according to Netcraft survey from May 2001 (www.netcraft.com/survey/) has slightly more than 60% of the world web server market (the second largest server, Microsoft IIS has about 20%). Another survey, the Internet Operating System Count of April 1999 (leb.net/hzo/ioscount) showed Linux as the largest Internet operating system with 28% of total, and another open source program, BSD, as the third largest program with 15% of the market (second place was taken by Microsoft Windows).

After a wave of enthusiasm in 1998 and 1999, when many companies decided to make open source related services and products an important part of their business, 2000 brought large losses to open source businesses (see e.g. *The Economist*, April 12th, 2001). These losses are often attributed to excessive initial expectations of businesses. It still remains to be seen whether this is a breakdown of open source as a business strategy, or a temporary crisis shared with the rest of the IT sector.

3 Stylised facts about open source software

This section contains some stylised facts about open source software, extracted from the literature about the subject. To learn more, see e.g. DiBona, Ockman, and Stone (1999), or Raymond (2000a-c).

3.1 Development

Typically, before a project is started, the basic program and the idea what it eventually should do, are already there. The first author places the basic code on the Internet - on a special site, like SourceForge, or a mailing list for programmers - with an invitation to co-operate on its further development. Most commonly, the first author is one person (Linus Thorvalds of Linux, Eric Allman of Sendmail) or a software company who decided to open the source of one of its programs (Netscape Mozilla). Often the original developer continues to manage the project, dividing it into smaller parts that can be developed independently and deciding about incorporating changes into the program.

Most projects are divided into modules, with a leader at the head of each module. The possibility of modular development is crucial, because it makes independent work on different parts of the program possible. The leadership of whole the project can be in the hands of one person, as is the case with Linux (Linus Thorvalds), or Sendmail (Eric Allman), collective as in Apache, rotational as in Perl. The initial experience with the project helps to establish credibility among programmers, which is crucial given that most of the work is done by volunteers. The programmers must believe that the project has a potential, that its leader(s) act not only in their own interest but also in the interest of the community and that they are competent. Hence, they must believe in its success and that they will have a share in it (see Lerner and Tirole, 2000).

Since reputation among peers seems to be an important incentive to provide high quality contributions, the project leaders have to make sure that the programmers' efforts are recognised. It sometimes happens in the form of listing the names of the most important contributors in the program itself. SourceForge ranks the contributors according to the quantity and quality of their work, where the ranks are given by other programmers and each rank is weighted according to the rank of the person who gave it. Because of the fact that development is taking place in an incremental way, the public good nature of the contributions and the importance of peer review and reputation, open source way of developing software is sometimes compared to the way in which science evolves.

3.2 Open source licenses

Most of open source software is protected by one of open source licenses, which differ in restrictions that they impose on users. The oldest of them, GPL of the Free Software Foundation, requires that all the changes made to the original code are made public as well, and that the GPL-ed software is distributed only with software covered by the same license. Because of the last requirement, this is a so-called "viral" license. On the other hand, the Open Source Definition allows keeping the changes private, although it does not allow anyone to appropriate the existing code. There are also many other types of licenses, some designed by companies who opened the source of the code but wanted to retain some control over it (e.g. Mozilla license of Netscape). An overview of open source licenses can be found on the website of Open Source Initiative (opensource.org).

3.3 Advantages and disadvantages to users

IT magazines and web sites cite many opinions of users and developers about currently available open source software and its advantages and disadvantages over commercial software. Those most commonly mentioned are listed below.

Lower price

The code of the open source software is essentially free. Making the program usable can require some additional work, like selecting and assembling different modules, which can be done by the user him/herself or by a specialised distributor. In the latter case, the ready version is often available for free on the web site of the distributor (e.g. Red Hat Linux), and even if this is not the case, open source software is likely to be cheaper than its commercial substitutes. On the other hand, this is not an exclusive feature of open source: many firms make a version of their product free without opening the source (for instance to make the quality of the product known and create demand for more sophisticated versions).

Higher reliability

It is a common opinion that many open source programs are better than their commercial counterpart, especially that they are more reliable and contain fewer bugs. This is supposed to follow from the fact that the code is examined by many programmers, which makes it more probable that a bug will be found and that someone will know a solution: the saying goes "given enough eyeballs, all bugs are shallow" (see e.g. Raymond, 2000b). The diversity of experiences, talents and interests makes sure that there is always someone who will find a given problem interesting and/or easy to solve. In other words, given that programmers are heterogeneous with respect to the costs of development, there will always be someone who will have a small (or even negative) cost of solving a given problem or making a given enhancement.

Possibility of customising and independent bug fixing
Many users find it an advantage that they can make changes to the program in order to make it more suited for their own needs or in order to fix bugs. This seems especially important for sophisticated users, but users without knowledge about IT can profit from it as well, by being able to hire competitive firms who can customise the program for them.

Avoiding "vendor lock-in"
Since everyone with enough expertise can understand the way the program works and make changes to it, the consumers can avoid being forced to hire the monopolistic owner of the code to provide technical support, make changes and enhancements. Hence, the user avoids being locked-in with one supplier.

Availability of support after the original creator abandons the project
The proponents of open source software argue that even if the author of the program does not provide support any more, the openness of the code makes sure that there is always someone who is able to understand the way the program works and provide support. This is in contrast to commercial programs, which are not supported any longer after the developer goes bankrupt.

More sceptical opinions about open source software can be heard as well. Some of the most feared disadvantages are:

Lack of an agent responsible for development, quality or support
It is sometimes thought that since commercial software producers are legally responsible for their product, there is more certainty about its quality. It seems that in practice legal suits on the ground of insufficient software quality are very rare. However, the mere possibility of such a suit may be enough to make sure of a certain minimum quality of commercial software. A counterargument is that it can be easier to find support for an open source program than for a closed one abandoned by the owner.

Forking and lack of compatibility between versions
The decentralised way of developing and distributing open source programs may lead to the emergence of many imperfectly compatible versions. The differentiation may happen already during the development of the program, during its distribution or during the adaptation for the users' needs. The first case is known as forking and occurs when the differences of opinion about the direction of the project lead to a division into separate projects, which develop further

independently. At the distribution level, many versions made by different distributors or users themselves may lead to imperfect compatibility, which reduces the value of program. There seems to be a danger of such user/distributor level fragmentation for Linux (see Appendix).

Incompleteness and poor integration

Some people argue that because open source software is developed in a voluntary and uncoordinated way, less attention is devoted to less attractive programming tasks, for instance documentation (on the other hand, the missing parts could be provided commercially by firms). That may result in incompleteness of open source programs (see e.g. Johnson, 2000). Also, the largely independent development of modules can lead to their poorer integration into one coherent program.

Lower user - friendliness

It is sometimes argued that open source software has a high "nerd - level", thus that it requires more knowledge on the part of the users. This may be a disadvantage for many users with poorer computer and programming skills.

Poor availability of complementary goods

For many types of software, like operating systems, the availability of complementary software is crucial. At this moment, however, the variety of applications that work with open source operating systems is significantly lower than that of the applications for Microsoft operating systems.

4 Impact on competition

This section analyses the impact of open source on competition, with a focus on prices and entry, as well as the investment in quality, the choice of product differentiation and compatibility which is motivated by entry deterrence or accommodation decision. The analysis begins with delineating the market in subsection 4.1, which also contains an overview of software markets characteristics under the assumption that all software is proprietary (that is, source codes are the property of commercial firms). Since strategic interaction between producers involves influencing each other's incentives, before proceeding to the study of competition one needs to know what motivates open source programmers to contribute to the development. An overview of these incentives, based on Lerner and Tirole (1999) can be found in subsection 4.2.

In subsection 4.3 the taxonomy of business strategies developed by Fudenberg and Tirole (1984) is applied to an analysis of entry deterrence and accommodation. The following instruments of deterrence and accommodation are taken into account: investment in quality and complementary goods, the choice of differentiation from the competitor and compatibility. To study the impact of open source, I compare the market outcomes in four situations: both the incumbent and entrant are proprietary, the incumbent is proprietary and the entrant open source, the incumbent is open source and the entrant proprietary, and both competitors are open source.

4.1 The market

The relevant markets are markets for software that can be separated from the hardware on which it is run, and which is universal enough to be of value to different users. In other words, we only consider markets for *packaged software*. Hence, we are not interested here in software which is an integral part of electronic equipment *(embedded software)*, for instance of mobile phones, or software which is developed for purposes of a specific company, either in-house or by a specialised firm. The markets for these types of programs have different characteristics and should be studied separately.

According to the function, one can distinguish the following types of packaged software (classification based on IDC, 1999).

Applications
This is software used by final users, consumers or firms, to perform the desired tasks
Programmer Development Tools: Software which developers use to make and implement software.
System Infrastructure Software: This group includes operating systems, needed to run applications.

In general, only applications have a value to final users. However, they have to be used in a system that contains, at minimum, also hardware and an operating system. Moreover, users usually need certain skills to be able to use programs. Hence, availability, price and quality of other components of the system are important for the value of any particular program. This is one of the sources of network effects in software markets, described in the next subsection.

It is a commonly known feature of markets for information goods that they tend to be dominated by one standard (see e.g. Shapiro, Varian, 1999). This is a result of on the one hand the supply side economies of scale - a large cost of producing the first copy and a very low marginal cost of additional copies, and on the other hand by the frequent presence of network effects on the demand side (also known as demand-side economies of scale). As a consequence, information goods can be produced more efficiently by a monopolist, and consumers tend to choose the product that already has a large market.

These features are present in markets for standardised computer software. On the supply side, the development of a program requires a large investment of programming and co-ordinating effort, while the cost of making and distributing an additional copy is negligible, especially since the Internet can be used as a distribution channel. On the demand side, it is often beneficial to use the software that has a large number of users.[5] The network effects are sometimes direct, for instance in the case of text editors where it is important that people can exchange files. They can be also indirect, based on the availability of complementary goods and services. These indirect network effects arise under the following conditions

- The value of the product depends on the availability of complementary goods and services
- Industries producing complementary goods are characterised by economies of scale on the supply and/or demand side.

These conditions are clearly satisfied in software markets. It was already mentioned that the availability of complementary hardware and software is very important for the value of most programs. This is especially important in the case of infra structural software, whose value relies almost entirely on the availability of applications. For both types of complementary goods (especially software) supply and demand side economies of scale are large. It follows that if a program has a lot of users it is more likely that a lot of complementary software will be available at low prices.

These economies of scale make the software markets prone to monopolisation, although the indirect demand side economies of scale have this effect only if the complementary goods and services are not fully compatible with all competing products, (otherwise all the competing products belong to the same network and create the same network benefits). Examples include

[5] For a discussion of network externalities, see e.g. Katz, Shapiro (1985).

in the first place Microsoft Windows and Microsoft Office. Programs offered by different developers exist in the market to the extent that they are horizontally or vertically differentiated, but even then large asymmetries between the market shares can occur (see e.g. Economides and Flyer, 1997). Given this monopolisation, which in the presence of economies of scale may be also socially optimal, the performance of the market can be better measured by entry possibilities than by the market shares.

4.2 The incentives to invest in an open source program

In order to study the behaviour of open source developers in the market, one needs to know their motivation to invest in the development. They are not motivated by profits from the sales of the program, as commercial firms, and systematic empirical studies of their incentives are lacking. Lerner and Tirole (2000) make an attempt to identify the most important incentives of open source developers. Their findings are summarised in this subsection.

Open source programs are usually developed by volunteers, with one group of volunteers managing the project and others sending in contributions. In some cases firms engage in the development as well, especially producers of complementary goods. Knowing the benefits of contributors we can try to establish what determines the total investment in an open source program.

According to Lerner and Tirole, a programmer contributing to a project, whether it is proprietary or open source, gets two types of payoffs:

(i) Immediate payoff, which includes:

(a) monetary compensation (only in commercial firms)
(b) private immediate benefit: solving problems that emerge during own use of the program
(c) opportunity cost of time

(ii) Delayed payoff.

Let me look at each of these components in more detail.

(i) Immediate payoff

(a) Financial compensation is usually only offered for contributing to commercial projects. There are, however, exceptions: some companies assign programmers to work on open source. In that case, it is the firm takes the role of a voluntary contributor who weighs private immediate and delayed benefits against the salary of the programmer.

(b) Private immediate benefits seem to be an important motivation to contribute. Typically, contributions are made by users of the program who want to increase its use value by removing bugs or adding extensions. Once a patch (a solution to a particular problem) has been made, the cost of making it available on the Internet is negligible. There is, of course, a free-rider problem here: everyone would prefer that others, rather than himself/herself, invest.
(c) The opportunity cost of time depends, among others, on possibilities to earn money elsewhere in the time spent on contributing and on the disutility of (or pleasure from) the programming. Lerner and Tirole give some reasons why the cost of programming may be lower for open source than proprietary programs; the most important one is the "alumni effect". If the code is available to everyone, it can be used for learning purposes, which makes it easier to work on it in future. This effect should be stronger if the program is more popular.

(ii) The **delayed payoff** includes a reputation gain. By contributing, a programmer signals his/her ability and willingness to do programming work (low cost of effort). The reputation can be a reward in itself ("ego gratification incentive"), or it can lead to higher wages in the future ("career concern").[6] The signalling incentive is likely to be stronger if the (expected) audience is large (the number of users, especially of those who are programmers themselves and are able to assess the quality of contributions). As an evidence, Lerner and Tirole note that most of contributors are users themselves and that open source project managers and participants are careful to recognise the effort of contributors. They also observe that some prominent open source developers did indeed get lucrative positions in the software industry.

From the point of view of our later analysis it is interesting whether the total investment in an open source project increases or decreases with the number of developers. Given the probability that an individual programmer invests, the chance that a problem gets solved is larger. On the other hand, the free-rider problem becomes more acute as the number of (potential) developers increases. An attempt to solve this ambiguity has been made by Johnson (2001), who studied this problem formally and concluded that the total investment increases with the size of developer base. Taking into account the reputation effect, Lerner and Tirole also conclude that a project will attract more investment if it has many developers.

The number of developers may be correlated with the number of users, but it need not be the case. One can expect it in the case of software that is mainly used by computer programmers, for instance system software and software development tools. For open source consumer applications, on the other hand, the correlation between the consumer and developer base is likely to be weaker.

[6] Interestingly, the incentives to contribute to open source may be larger if a larger proportion of programs in the market are commercial. The reason is that the firms' demand for programmers and the future returns to reputation may be larger in that case.

4.3 Entry deterrence and accommodation

Open source programs, just as proprietary ones, tend to monopolise markets. This follows from the network effects on the demand side and may be reinforced by the supply side network effect, which makes projects with many developers more attractive to invest in. Hence, this basic feature of software markets is not likely to change as open source becomes popular. This is not necessarily bad, since the price of open source programs is zero, and since the economies of scale and network effects make a monopoly more efficient. However, open source may have an impact on the market performance by influencing barriers to entry, thereby affecting prices and the possibilities of more efficient producers to enter the market.

This section applies the taxonomy of business strategies developed by Fudenberg and Tirole (1984) to an analysis of the impact of open source on competition the market. In particular, the following questions are addressed.

- How will a proprietary incumbent behave in the presence of an open source entrant, and how does it compare with the strategies used in the presence of a proprietary one?
- Is an open source program more or less likely to enter, and what kind of open source programs will enter most often?
- What are the barriers to entry if the incumbent is open source, and does an open source entrant face higher or lower barriers than a proprietary one?

The analysis begins with a short summary of the taxonomy of Fudenberg and Tirole in subsection 4.3.1. In subsection 4.3.2 entry of a proprietary and an open source program is analysed under the assumption that the incumbent is proprietary. In subsection 4.4.3. the case of an open source incumbent is explored. The analysis is illustrated with the examples of open source projects taken from the Appendix.

4.3.1 Classification of business strategies

Fudenberg and Tirole (1984) consider a two-period model with two competitors, an incumbent and an entrant. In the first period, the incumbent makes an investment, which remains sunk, i.e. irreversible, for two periods. In the second period, both firms compete in the market. By choosing the investment level, the incumbent can influence the competitor's actions: the decision to enter, or the price or quantity in the next period. Depending on the impact that the investment has on future competition and the profits of the entrant, it may be optimal to invest more (overinvest) or less (underinvest) than in the absence of a strategic effect.

Since capacity constraints in the production of software are practically absent, the assumption of price rather than quantity competition between firms seems reasonable. Taking the decision to deter or accommodate entry as given, different investment strategies are available:

(i) Entry deterrence

Suppose that the incumbent wants to deter entry. Then, one of the following strategies can be used, depending on the situation:

- Overinvestment *(Top Dog)* is optimal if investment decreases future profits of the entrant
- Underinvestment *(Lean and Hungry Look)* is optimal if investment increases future profits of the entrant

(ii) Entry accommodation

- Overinvestment *(Fat Cat)* is optimal if investment increases the future price of the entrant
- Underinvestment *(Puppy Dog)* is optimal if investment decreases the future price of the entrant

Below, this taxonomy is applied to analyse entry when either the entrant or the incumbent is open source, and to compare it with entry when the entrant and/or the incumbent is proprietary. To make the analysis easier to follow, in the case of a proprietary incumbent I choose to focus on a few types of competition-influencing decisions that seem to be the most relevant in the context of software markets. The following kinds of investments are considered:

Investments that increase the value of the program for all users

- Investment in quality
- Investment in the development of complementary software or encouraging its development by other firms. That can be done by lowering other firms' costs of developing complementary software, e.g. by creating documentation or easy to learn programming languages, making good application interfaces, providing training and support to programmers. This strategy was followed by Microsoft, who encouraged the creation of applications for Windows. The entrant faces than a so-called "applications barrier to entry".
- Investing in the consumer base, e.g. by setting initially a low price or by facilitating acquiring of complementary skills. The latter can be done by making the program easy to learn and user-friendly, distributing handbooks, providing training, technical support via the Internet etc. Once a large consumer base is created, consumers face switching costs when adopting another program.

For the sake of concision, I will sometimes refer to all these investments as an investment in "quality".

The choice of differentiation from the competitor's product.
The decision about the degree of substitutability between the programs has an impact on the price and market share of the competitor. Hence, it can be used as a strategic instrument. To study product differentiation, I will most of the time use the results of a Hotelling[7] horizontal differentiation game with quadratic transport costs and consumers uniformly distributed on the interval (0,1).

The choice of the degree of compatibility with the competitor's product.
Network externalities in markets for information goods follow, for a large part, from availability of compatible complementary goods. That makes compatibility an important strategic instrument. For the purposes of this paper, the degree of compatibility between two programs is defined as the proportion of complementary goods and skills that can be used with both of them. Since an entrant does not have its network yet, only backward compatibility is relevant, i.e. compatibility of the entrant's program with that of the incumbent.

In what follows, I begin with the discussion of entry when the incumbent is proprietary in subsection 4.3.2. The next subsection, 4.3.3, proceeds with an open source incumbent.

4.3.2 Proprietary incumbent

Here I assume that the dominating program belongs to an incumbent firm who controls its source code. It is also assumed that the decision to deter or accommodate entry is given; at the end of the section the incentives to deter and accommodate are compared. I begin with an analysis of entry deterrence in *(i)*. First, a benchmark is set with a description of strategies used towards a proprietary entrant. Subsequently, entry deterrence of an open source program is discussed, and compared with the benchmark case. This will allow for identifying the conditions under which deterring entry of open source is most costly, and to compare the costs of deterring entry of proprietary and open source entrants.

An analogous analysis for entry accommodation can be found in *(ii)*. In *(iii)* one can find a discussion of possible incentives of the incumbent to accommodate or deter entry. Comparing these incentives leads to some predictions about the types of open source programs that are most likely to enter.

The conclusions at different stages are illustrated with the case of open source programs from the Appendix. One of the most interesting examples of competition between a proprietary

[7] Originally proposed by Hotelling (1929). See also e.g. Tirole (1997).

incumbent and an open source entrant is the case of Microsoft Windows and Linux. Most attention is therefore devoted to that case.

(i) Entry deterrence

(a) Proprietary entrant

Suppose that a proprietary incumbent wants to deter entry of a proprietary competitor. To do this, it has to make sure that the entrant makes negative profits after entry. This can be achieved by making in advance (i.e. before entry) investments that will decrease the future market share or the price of the entrant. Let me now consider the use of different types of investments as strategic instruments.

Investment in quality, complementary goods and consumer base

On the one hand, these investments have a direct negative effect on the entrant's profits by decreasing its demand. On the other hand, they have a positive strategic effect: the increased demand for the incumbent's program increases its future price. This allows the entrant to increase its price and profits.

If the direct negative effect on the entrant's demand dominates, the optimal deterring strategy is a high investment, or "top dog". The incumbent makes sure that it has a large advantage over the entrant, so that few consumers will want to switch. If the positive effect of higher prices dominates, the incumbent can better apply the "lean and hungry look" strategy of low investment in order to commit to more aggressive pricing in the future. One may expect that in software markets, in the presence of network effects, sellers are more likely to make use of higher demand to increase their sales rather than prices. Indeed, the prices of software have been falling in the last years, in spite of a constant increase in quality. Hence, it seems that a demand increasing investment is less likely to lead to higher prices, which implies that the direct negative effect on the entrant's demand will more often dominate. Thus the "top dog" deterrence strategy, under which the incumbent invests a lot in quality, can be expected to be more frequent.

Differentiation

In the Hotelling model described earlier, placing one's product closer to that of the competitor decreases the competitor's market share and price. Hence, introducing a product similar to that of the entrant deters entry (this is also known as pre-emption). In terms of the taxonomy, this corresponds to the "top dog" strategy (moving towards the centre of the market can be treated as an investment in increasing the market share).

Compatibility

Higher compatibility increases the entrant's demand, but it may also intensify price competition after entry, because the programs become better substitutes. If the former effect dominates, an entry-deterring incumbent should avoid compatibility. The lack of compatibility will not allow the entrant to make enough sales to cover the costs of entry (this corresponds to the "top dog" strategy). On the other hand, if the effect on prices dominates, the incumbent may deter entry by becoming compatible with the entrant, which will intensify the future price competition and decrease the entrant's profits (corresponds to the "lean and hungry look" strategy).

(b) Open source entrant

Since open source developers are motivated by different benefits than commercial firms, the strategic effects of the incumbent's actions will also be different. To deter entry, the incumbent's investment must influence the number of developers or the individual developer's incentives to contribute (one of which is the number of developers). It can also try to discourage the producers of software or hardware to make products complementary to the open source. Below, I consider entry deterrence strategies based on diminishing the developers' incentives as identified by Lerner and Tirole (2000). After describing these strategies, I try to determine under what conditions they are likely be the most effective (least costly). Next, the entry deterrence strategies towards a proprietary and an open source entrant are compared, as well as market outcomes in both cases.

Deterring entry of an open source program relies on decreasing the motivation of its developers. To do that, the incumbent can attempt to decrease the (relative) use value of the open source program, their reputation or career concern, or increase the costs of contributing.

Some examples of such strategies are given below.

- The incumbent can decrease the use value of investment in open source by increasing the relative value of its program to the potential developers. The developers may then adopt the commercial program instead of the open source. The incumbent can increase the value by lowering the price of its program, increasing its quality and availability of complementary goods and its market share, as well as preventing compatibility and decreasing product differentiation.
- The incumbent could increase the opportunity cost of contributing by hiring open source programmers and paying them high salaries for programming for the incumbent rather then for the open source project (but this increases the "career concern" motivation - see below). The opportunity cost can also be lowered by creating an own "alumni effect" - e.g. by a free distribution of training programs similar to the proprietary one.
- The reputation gain could be decreased by reducing the "ego gratification effect", e.g. by trying to publicly discredit the skills and motivation of the contributors. That may be a reason for

Microsoft's verbal attacks against the open source movement. However, the effectiveness of such actions seems doubtful. Another possibility may be decreasing the market share of the entrant, which may decrease the audience to the programmer's performance.

- The "career concern" motivation could be decreased by offering lower salaries to programmers, or refusing to hire those who are known to contribute to open source programs. However, this would be in conflict with the earlier mentioned strategy of increasing programmers' opportunity costs. Moreover, since companies have an incentive to hire best programmers, it might be difficult to commit credibly to such a policy.
- Commercial developers can be discouraged from writing complementary software by decreasing the market share of the open source program and by making programming for the incumbent's program more attractive.

Note that the first group of strategies coincides with those used against a proprietary entrant. However, while the entry deterrence of a commercial entrant relies on decreasing its future sales or price, this is not the case with open source. The benefits of the open source developers depend less on the market share than the profits of a commercial firm, and therefore decreasing the future market share of an open source entrant will be less effective. Similarly, the benefits of open source developers do not depend on the price of their product, which always remains zero. Hence, any entry-deterring strategy based on intensifying future price competition does not work. The price of the commercial alternative may affect the incentives of developers, but this is only one of many factors. On the other hand, providing a high quality alternative for the open source may be effective as an entry deterrent in a different way if its developers will be encouraged to adopt the commercial product instead of the open source. In particular, to deter entry the incumbent may want to focus on satisfying the needs of those users who are most likely to become developers, that is sophisticated users.

Knowing the entry deterrence strategies used against both types of entrants, one could try to compare the costs of deterring their entry. This, however, turns out to be difficult, because it is not known how sensitive the incentives of open source developers are to the entry -deterring strategies as compared to the incentives of commercial firms. On the other hand, it is possible to point to the factors likely to make the entry of open source more difficult to deter. The reputation concern should be more difficult to influence if it does not depend on the number of users. Focussing further on strategies presented under the first bullet above (which diminish the immediate benefits), one can conclude that entry deterrence is more difficult if it is difficult (costly) to provide a good alternative to the open source program. This may be the case if:

- Open source developers have better information about the tastes of users.
- Some users have a strong valuation for the features in which open source has inherent advantages (i.e. which are costly for a proprietary firm to introduce) and a lower valuation for those in which proprietary firms tend to be better.
- Decreasing the future market share for the open source has little impact on the investment in the open source program. That is the case if the developers do not care about the number of users (e.g. because the network effects are relatively unimportant) or if the free-rider problem becomes much less strong when the number of developers decreases.
- The presence of complementary software is not very important
- It is difficult to prevent the entrant to achieve compatibility

(ii) Accommodation of entry

Suppose now that entry deterrence is not optimal (too costly). Then, the incumbent will choose the level of investment maximising profits given that entry occurs. The investment will have a direct positive effect on profits, but if the entrant is proprietary, it may also have a strategic effect through the impact on the competitor's price. If this is the case, the incumbent may want to invest more or less than the direct effect would imply in order to decrease the future price competition. I proceed now to the analysis of entry accommodation strategies.

(a) Proprietary entrant

Every action of the incumbent will have both a direct effect on profits and a strategic one, through the impact on the entrant' price. I again consider the different types of investment:

Investment in quality, in complementary goods, and in increasing of the consumer base

This type of investment has a positive direct effect on the incumbent's demand, and a strategic effect through the competitor's price, which consists of two components. First, the investment decreases the demand for the entrant's program, which also decreases its future price. Second, thanks to the increased demand the incumbent can set a higher price in the future, which allows the entrant to set a higher price as well. Hence, the sign of the whole strategic effect can be positive or negative. In the former case, an entry accommodating incumbent should become a "fat cat" by overinvesting, because that commits it to high future prices and encourages the entrant to set high prices as well. In the second case, the incumbent prefers to be a "puppy dog", that is to underinvest in order not to decrease the entrant's demand too much. In other words, it "bribes" the entrant by leaving it a part of the market, in exchange for a less aggressive price behaviour.

It was mentioned before that in software markets increased quality does not usually lead to higher prices. That suggests that the strategic effect of an investment is more likely to be negative, and the optimal entry accommodation strategy will most often be "puppy dog":

underinvestment in quality in order to leave the entrant some profits and in this way discourage it from starting a price war.

Differentiation

In the Hotelling spatial differentiation game, entry accommodation involves increased product differentiation. By moving away from the entrant, the incumbent gives up a part of the market, but it also decreases the future price competition. Thus, this corresponds to the "puppy dog" strategy: the incumbent leaves a part of the market to the entrant, who then finds it less worthwhile to price aggressively.

Compatibility

A higher compatibility increases the value of both products to consumers, but it decreases the advantage of the incumbent over the entrant. Hence, the direct effect on the incumbent's profits can be either positive, or negative. It is more likely to be negative if the market is already saturated, because then the increase in the demand of one producer can only happen at the expense of the demand of its competitor. On the other hand, in markets that still have a large potential to grow, compatibility may also increase the demand of the incumbent.

The strategic effect of the increased compatibility may go either way as well. On the one hand, a higher compatibility increases the entrant's price due to increased demand, but on the other hand it decreases its price by intensifying price competition. If the net impact of compatibility on the entrant's price is negative, an accommodating incumbent should choose its lower level in order to increase prices in the future ("puppy dog" strategy of giving up a larger market for a lower price competition). On the other hand, if the positive effect on the entrant's price dominates, allowing more compatibility may be better.

In markets with strong network effects, increased demand is likely to lead to higher sales rather than to a higher price. Hence, in software markets, especially those where network effects are relatively strong, the strategic effect of compatibility on prices is more likely to be negative. That would suggests that in order to keep the post-entry prices high, the incumbent should avoid compatibility more than the direct effect on profits would imply.

(b) Open source entrant

Since an investment has no strategic effect on the entrant's price, one cannot speak of entry accommodation in the sense of over - or underinvestment, but only in the sense of "not deterring". The incumbent maximises its profits taking into account the direct effect alone. Thus, I describe entry accommodation by comparing it to the entry deterrence. Afterwards, I compare the accommodation of an open source and a proprietary entrant. Let me consider the different types of investment:

Quality, complementary goods and the consumer base

When deterring entry, the incumbent will never choose a higher level of quality, complementary goods or market share than when accommodating. If the profit-maximising level is sufficient to deter entry, it is chosen; else, the quality must be higher. The entry-deterring levels will be much higher if:

- the investment is costly and the demand reacts poorly to increased quality (the number of complementary goods, consumer base), because then the profit maximising level will be low
- the investment has little effect on the developers' incentives, because then the minimum entry-deterring level is high

Since we focus on deterring entry by decreasing the relative use value of the open source, the same factors which make it more difficult to discourage open source developers (identified when discussing entry deterrence) make it difficult to influence the demand. Hence, the more costly entry deterrence is, the more positive its impact on investment in quality.

Differentiation

In the Hotelling game, if entry occurs, a profit maximising incumbent differentiates maximally. That is, it spends relatively many resources on those features of the program which are underdeveloped in the competing program. This is the opposite to what should happen under entry deterrence: to deter entry, the incumbent locates itself as close to the competitor as possible, which means trying to improve those features in which the entrant has an advantage.

Compatibility

Entry deterrence requires complete incompatibility. This may also be the case under entry accommodation, if the direct effect of compatibility on the incumbent's profits is negative (this is more likely if the market does not have much potential to grow and if the programs are close substitutes). On the other hand, in markets with strong network effects and in which there is still much potential to grow, some compatibility may be optimal. It follows that entry accommodation leads to no less, and perhaps more compatibility than deterrence.

This comparison of entry deterrence and accommodation strategies can be applied to an analysis of the competition between Microsoft Windows and Linux. On the one hand, the entry deterrence case may seem irrelevant, since Linux already entered. On the other hand, because of a fast technological progress in software markets, one may assume that both Microsoft and Linux have to "enter" continuously with new versions of the programs. Thus, speaking of "entry deterrence" in that case seems justifiable.

Note first that Microsoft is unlikely to make Windows compatible with Linux even when accommodating entry. That is the case because the market for operating systems does not grow fast enough any more (at least in Western countries) and Linux seems to be close enough a substitute for Windows to eat away its profits if compatibility is achieved. Hence, the direct effect of compatibility on Microsoft's profits is likely to be negative.

How about other investments? To deter entry, Microsoft should increase the demand for Windows by investing in higher quality, especially in those aspects of quality in which Linux now has an advantage (decrease differentiation). For instance, it is said that Linux (and in general open source software) is more stable and reliable than Windows. Hence, this strategy would involve spending more resources on increasing stability and reliability. Another example is allowing more adjustability of Windows for users' needs, in order to match another advantage of Linux, the openness of its code. Microsoft should also encourage the production of applications compatible with Windows and not with Linux.

As regards entry accommodation, the most important aspect that distinguishes it from deterrence seems to be differentiation from the competitor's product. That means identifying the weak points of Linux and making sure that the analogous features in Windows are of a high quality. For instance, Microsoft could pay extra attention to user- friendliness and availability of manuals and technical support, which is said to be a relative weakness of Linux. Another weakness of Linux - which is the negative side of one of its advantages, adjustability - is the presence of its many versions, which are not always compatible with one another. Instead of trying to increase its own adjustability, Microsoft can offer perfect compatibility between its different products and win the consumers for whom it is more important than the possibility to adjust the features of the program.

Is there some evidence that Microsoft is engaged either in entry accommodation, or entry deterrence? Avoiding compatibility of its Windows applications (like Microsoft Office) with Linux, which indeed happens, is probably optimal under both strategies. Investing in quality, complementary software and consumers base has always been taking place, and it is difficult to say whether it increased after Linux appeared. Can one observe increased differentiation or just the opposite, pre-emption? The aspects of quality which are underdeveloped in Linux (availability of support, documentation, user-friendliness) have always been paid attention to at Microsoft. As for matching the strong points of Linux, there is some evidence that Microsoft is taking some actions to match the advantages of Linux; recently, largest consumers have been given an access to the source code, although they are not allowed to change it (see *The Economist, May 12th, 2001*). But, all in all, I do not possess enough information to claim that Microsoft is engaged in entry deterrence or accommodation.

To make the analysis of the accommodation complete, let me compare the investment levels when accommodating the two types of entrants.

Investments in quality, complementary software and the consumer base.

These investments have a direct positive effect on the incumbent's profits in the presence of both entrants. However, the strength of this direct effect may be different. To see that, note first than given the investment level, the prices in the market will be lower when the entrant is open source, because its price is zero. That influences the profitability of the incumbent's investment. If this direct effect increases with the price of the competitor, it will be more positive when the entrant is proprietary (this is for instance the case when demand is linear[8]). That would imply a higher investment in the presence of a proprietary entrant.

The sign of the strategic effect on the price of the proprietary entrant is ambiguous. If it is positive, than combined with the higher direct effect on profits it would result in higher incentives to invest in quality when the entrant is proprietary. However, as already mentioned when discussing entry accommodation, there seem to be reasons to believe that in software markets the strategic effect is negative. If that is the case, and if the conjecture that the direct effect is more positive in the case of proprietary entrant, is true, it is ambiguous in which case the incentives to invest in quality are higher.

Differentiation

Given the price of the entrant, higher differentiation on the one hand increases the price that the incumbent can charge, but it decreases its market share. In the Hotelling game with uniformly distributed consumers and quadratic transportation costs, the total direct effect of increased differentiation on profits is positive and decreasing with the price of the competitor, thus more positive when the entrant is open source.

In addition, higher differentiation increases the future price of a proprietary entrant, which has a strategic positive effect on the incumbent's profits. That means that it is ambiguous which type of entrant induces more differentiation. On the one hand, the direct effect may be stronger in the presence of open source, but on the other there is no strategic effect. In the Hotelling game the total incentives to differentiate are stronger if the entrant is proprietary. However, it is difficult to say whether it will be more generally the case.

Compatibility

As written earlier, the direct effect of increased compatibility on profits can be either negative or positive, and it depends on how much it increases the total demand for both sellers' products. It is more likely to be positive if network externalities are strong, if the market is new, with a lot of potential to grow, and if the programs are not very similar. It also seems that it should be more

[8] Let the incumbent’s demand be $D_1 = q - bp_1 + dp_2$, where q is quality. Assume no unit production cost (a realistic assumption in the case of software) and no cost of quality (since it does not depend on p_2, this omission does not matter. Then, optimising profits with respect to p_1 gives $Đ_1 = (q+dp_2)^2/4$. The direct effect of increased quality on profits is $(q+dp_2)/2$, which is increasing in p_2.

negative when the price of the competitor is low, thus more negative when the entrant is open source. On the other hand, the incumbent may have a strategic reason to avoid compatibility with a proprietary entrant more than it is justified by the direct effect alone. Hence, it is again ambiguous whether the incumbent will avoid compatibility with open source more than with a proprietary entrant.

It seems that on the basis of the above analysis, one cannot make a clear comparison between the accommodation strategies used towards proprietary and open source entrants. The levels of quality, differentiation and compatibility chosen by the incumbent can be higher either in one, or the other case.

(iii) Incentives to deter and accommodate entry.

The decision to deter entry of an open source entrant will depend on the one hand on the costs of deterrence, and on the other hand on the profits after entry. An open source entrant lowers prices in the market more than a proprietary one, other things being equal, and I found no reasons to believe that the incumbent will choose systematically lower or higher levels of quality, differentiation or compatibility when facing an open source entrant. Thus, one might expect higher incentives to deter an open source entrant. On the other hand, it is ambiguous whether deterring entry of open source is more or less costly. Hence, the question whether open source programs will enter more easily than proprietary ones, cannot be answered unambiguously.

On the other hand, one can try to determine the conditions under which deterrence or accommodation of open source will be optimal. The conditions under which the costs of deterrence are the highest have already been described when studying entry deterrence. To compare profits under entry accommodation and deterrence, I focus on the aspect of the accommodation strategy that distinguishes it most clearly from entry deterrence: high differentiation from the entrant's program.

In the Hotelling horizontal differentiation model, the profitability of this strategy depends on the location of the open source competitor. It is the lowest if the entrant is located in the middle of the market, and the highest if it is located at its boundaries. Thus, differentiation is most profitable if the open source program is a niche product, which matches the tastes of a small group of users only. If the entrant is attractive for average users as well, the incumbent loses a large part of the market share if it decides to allow entry. Thus, entry is more dangerous for the incumbent if the open source entrant is attractive for typical consumers.

On the other hand, entry deterrence is more costly if the open source program has some features which are attractive for developers, and which are difficult to imitate by the incumbent. Comparing the benefits and costs of deterrence, one gets the following typology of open source entrants and the corresponding strategies:

• Costly to deter, attractive mainly to developers	- accommodate
• Costly to deter, attractive both to developers and average users	- ?
• Easy to deter, attractive mainly to developers	- ?
• Easy to deter, attractive both to developers and average users	- deter

If the cost of deterring entry is large, and the open source program stays in a market niche after entry, the incumbent prefers to accommodate. Then, both programs will coexist. On the other hand, if the program has features which could give it a large market share after entry at the expense of the incumbent, but which are easy to imitate, entry deterrence is the most likely outcome. Finally, if both costs and benefits are large, or small, the outcome is not clear: either entry deterrence or accommodation may result.

Applying this typology to the Microsoft/Linux case, what can we say about the optimality of deterrence or accommodation? Linux has features which are quite attractive for some users, and which are difficult to imitate because they are inherent to open source, like the openness of the code, which allows the users to make their own adjustments. Other features, like stability and reliability, are also often argued to be better developed in open source programs. Hence, one can assume that deterring Linux is costly. The benefits from deterring deter depend on whether the advantages of Linux are valued by a small group of users, and disadvantages (lower user friendliness, lack of support and documentation, lower standardisation) by most, or vice versa. At this moment, Linux still seems to be a niche program. In the long run, the profitability of accommodation will depend on whether the weaknesses of Linux will persist. That in turn depends on whether they are inherent to open source, or whether they are its "infant diseases" which will disappear as the movement matures. In the latter case, relying too much on the differentiation strategy may prove risky for Microsoft in the long run.

The typology can also be applied to characterise the types of open source programs that are most likely to enter. Given the incentives to accommodate, one should observe most entry in markets where a small group of users, who are also programmers, values the openness of the code or other inherent advantages of open source. Are the existing open source programs like that? Most selections of open source projects, including the one that can be found in the Appendix of this paper, are biassed towards large and popular programs, and therefore they are not good examples here. On the other hand, this might be an explanation a huge number of very small open source projects, used by a small group of users-developers: these might be programs filling market niches which proprietary firms do not find worthwhile to provide for.

The next group of open source programs, although a less numerous one, which one should observe, are those which gained a large market share from the competitor because their users' preference for open source was so high that entry deterrence by the incumbent was very costly. The preference for open source is likely to be larger in markets where most users are also programmers, because in these markets many users will often value the possibility to make own changes, as well as stability and reliability (especially if they are system administrators). The same users will often care less about the aspects of quality which are commonly less well developed in open source (user friendliness, availability of documentation and support). Indeed, most the open source programs which gained a large market share exist in markets in which a large proportion of users is sophisticated, for instance programming languages (Perl, Python), operating systems used by system administrators (Linux, Free BSD), or software used for communication or management of the Internet (Apache, Sendmail, BIND).

4.3.3 Open source incumbent

Open source developers do not really follow an entry "deterring" or "accommodating" strategy. They are committed to zero prices, and their investment in development or in complementary goods are most likely to be deprived of any strategic considerations; since the group of developers is large, each of them has only a small influence on total investments and on the actions and profits of competitors. However, it still makes sense to compare barriers to entry in a market initially dominated by an open source program with those that prevail in a market in which the incumbent program is owned by a firm. Since the barriers may be different for a proprietary and an open source entrant, these two cases are analysed separately.

(i) Proprietary entrant

To compare barriers to entry in a market with an open source and a proprietary incumbent, one should compare the quality, differentiation, compatibility and price chosen by an open source and an entry-deterring or accommodating incumbent.

Price

An open source incumbent has always a lower price than a proprietary one, since a proprietary incumbent cannot credibly commit to zero price in the long run, even if it deters entry. Hence, other things being equal, profits after entry are lower when the incumbent is open source.

Quality

Since it is difficult to compare the incentives of a proprietary firm and open source developers to provide quality, it is also hard to compare the levels of quality of an open source or proprietary incumbent. Quality may be higher in either case.

Differentiation

In terms of the Hotelling game, an entry-accommodating incumbent locates at a boundary of the market; an entry -deterring one locates in the middle. An open source program may be located anywhere in between (because its developers have no strategic considerations).
After-entry profits of the entrant (and therefore the incentives to enter) are the highest if the

incumbent program is located far from the centre (that is, if it satisfies the tastes of a small group of users only), and on the costs of entry.

Compatibility

Because of the openness of the code, it should be easier to achieve compatibility with the open source. Hence, the entrant can choose the optimal degree of compatibility from its point of view. When the incumbent is proprietary that is not always possible. Thus, open source should encourage more entry.

All in all, it is ambiguous whether an open source incumbent encourages or discourages entry more or less as compared to a proprietary one. It discourages it with its zero price, but encourages by allowing more compatibility. The incentives to enter depend also on whether the proprietary incumbent deters or accommodates entry, and on the location of the open source incumbent. The better the incumbent matches the tastes of average users, the more it discourages entry. That means that one should see less entry of proprietary programs in markets in which many users have a preference for inherent advantages of open source. According to previous analysis, these are also the markets in which open source is most likely to get a large market share after entry. That indicates that if open source dominates in a market, then many users may have a strong preference for open source, which in turn makes it difficult to enter with a proprietary program. That leads to a conclusion that one should observe less entry of proprietary programs in markets in which open source dominates, for instance the market for web servers (where 60% of the market belongs to Apache) or Sendmail (which serves 80% of the e-mail traffic).

(ii) Open source entrant

In this case, zero price is not a barrier to entry. On the other hand, the entrant may face other factors, which diminish the benefits of its developers:

- Large installed base of the incumbent. This may be a barrier if the number of developers or their benefits depend on the market share.
- Large developer base. An open source entrant has to compete for developers with the incumbent, who has a larger developer network and is therefore more attractive for programmers. On the other hand, being one of the first developers of a new project may bring higher reputation gains than making incremental improvements to the incumbent program.
- The availability of a free substitute decreases the use value of starting a new project
- The absence of complementary goods decreases the value of the new program if it is incompatible with the old one

- The incumbent is known by many programmers, which creates an "alumni effect" and makes it easier to contribute to.

It follows that the presence of an open source incumbent decreases the incentives to start a new project. Is this effect more or less negative than on a proprietary entrant? On the one hand, since the open source developers' benefits do not depend on the future price and less on the market size than those of proprietary firms, it should be more difficult to discourage them. On the other hand, a proprietary entrant can offer to users the features that are often less well developed in open source: user friendliness, good technical support and documentation, higher standardisation between versions. It seems that a proprietary entrant is more likely to enter if many users value those aspects of quality which are less well developed by open source. Otherwise, an open source project can enter more easily.

Finally, when are the incentives to start an open source project larger: when the incumbent is proprietary, or when it is open source? If many users value open source, the incentive may be larger when the incumbent is proprietary; on the other hand, a proprietary incumbent may have an incentive to deter entry, for instance by preventing compatibility, which is not the case when the incumbent is open source. Hence, the answer to this question is ambiguous. When only a few users value the inherent advantages of open source, an open source program is not likely to be an incumbent. Therefore, this situation is less relevant here.

4.3.4 Summary of the findings.

The main results from the preceding analysis are summarised in Tables 1-3. They answer to the three questions asked at the beginning of subsection 4.3: *What is the behaviour of a proprietary incumbent in the presence of an open source and a proprietary entrant? What kind of open source programs are most likely to enter? What are the barriers to entry when the incumbent is open source?*
Table 1 corresponds to the first question and summarises, and when possible compares, the incentives of a proprietary incumbent to choose the levels of quality, differentiation and compatibility when accommodating and deterring entry of a proprietary and an open source entrant. The table also compares the incentives to deter or accommodate an open source and a proprietary entrant.
Table 1: A summary and comparison of the incentives of a proprietary incumbent to select quality, differentiation and compatibility level when accommodating and deterring entry, and the incentives to deter or accommodate, when the entrant is proprietary and open source.[9]

[9] In the table, “stronger” means stronger than when the entrant is of the other type. “Possibly” means that the sign is ambiguous, but there are reasons to believe that it is like suggested in the table.

Table 4.1 Incentives of a proprietary incumbent

Entry accommodation	Proprietary entrant	Open source entrant
Incentives to invest in quality		
1. Direct effect on own profits	Positive, possibly stronger	Positive, possibly weaker
2. Strategic effect through the entrant's price	Possibly negative	Absent
Incentives to differentiate		
1. Direct effect on own profits	Ambiguous, possibly less positive/more negative.	Ambiguous, possibly more positive/less negative.
2. Strategic effect	Positive	Absent
Incentives to allow compatibility		
1. Direct effect on own profits	Possibly negative and weaker	Possibly negative and stronger
2. Strategic effect	Possibly negative	Absent
Entry deterrence		
Incentives to invest in quality		
1. Direct effect on the entrant's benefits	Negative	Negative
2. Strategic effect through price competition	Positive	Absent
	Total effect possibly negative	
Incentives to differentiate		
1. Direct effect on the entrant's benefits	Positive	Positive
2. Strategic effect	Positive	Absent
Incentives to allow compatibility		
1. Direct effect on the entrant's benefits	Positive	Positive
2. Effect through price competition	Possibly negative	Absent
	Total effect ambiguous	
Incentives to deter or accommodate		
1. Effect of entry on prices	Negative, weaker	Negative, stronger
2. Costs of deterring	Ambiguous	Ambiguous

As can be seen from Table 1, the relative incentives to deter the entry of proprietary and open source entrants are ambiguous. More can be said about the incentives to deter different types of open source programs. The typology and examples of programs fitting into different categories are summarised in Table 2, which summarises the answer to the second of the questions mentioned above.

Table 4.2 Incentives to deter or accommodate different types of open source entrants.

Open source entrant	Benefits from deterring	Costs of deterring	Examples
- Attractive mainly for a small group of developers - Its advantages are inherent to open source	Low	High	- Small projects at e.g. SourceForge.org - Linux?
- Attractive for many users - Its advantages are inherent to open source	High	High	- Linux? - Apache, Perl, Sendmail
- Its advantages can be easily imitated by a commercial firm	Low	Low	- Small projects at e.g. SourceForge.org
- Attractive for many users - Its advantages can be easily imitated by a commercial firm	High	Low	-

The last table, corresponding to the third question, compares the most important barriers to entry for proprietary and open source entrants in a market where the incumbent is either proprietary or open source. In each of the four cases those barriers are mentioned given which are relatively most important as compared to other cases.

Table 4.3 Barriers to entry

Barriers to entry	Incumbent	
	Open source	Proprietary
For proprietary entrants	- Low prices after entry - Possible preference of users for open source	- Possible active entry deterrence - Compatibility difficult to achieve
For open source entrants	- Large installed base of developers - "Alumni effect" - Availability of a free substitute	- Possible active entry deterrence - Possible preference of users for proprietary programs - Compatibility difficult to achieve

Before proceeding to the conclusions let me consider one more factor which seems to be important for the comparison of closed and open source. One of the often-mentioned advantages of open source above proprietary programs is that they can be adjusted to the users' needs. However, that may lead to too little standardisation as compared to the case when a proprietary program dominates. The following section explores the trade-off between standardisation and variety in the open source and proprietary cases.

5 Standardisation and variety

One of the problems faced by open source is insufficient standardisation. Different versions (distributions) of Linux, sold on CDs, are not always completely compatible with one another. These incompatibilities lower the value of the program by reducing the network benefits and discouraging programmers from writing applications for Linux. A related problem is forking, that is splitting open source projects into several incompatible programs that begin their own life. Though relatively infrequent, it reappears in the discussions about open source as a potential threat to its success.

According to the economic theory, standardisation is beneficial because it increases direct and indirect network benefits. In the case of software, standardisation facilitates exchange of files between users and encourages the development of complementary goods (software, books, and services). Another reason to limit the number of varieties is the cost of their introduction. On the other hand, heterogeneity of consumer tastes provides an argument for more variety. Hence, the consumers, firms and the social planner face a trade-off. This section discusses how this trade-off is resolved by the social planner, open source developers/users/distributors and a monopolistic seller of a proprietary program.

Salop (1979) in his circular city model shows that if varieties are chosen by different firms, there is too much differentiation from the social point of view. The reason for this is that profits from introducing a new variety follow partly from the increase of the product value to consumers, but also from "business stealing" from other firms. Hence, the incentives to introduce a new variety are too large from the social point of view. Navon, Shy and Thisse (1995) show that this result holds also in the presence of network externalities - provided that they are not too strong - even though they increase standardisation. Although network externalities increase standardisation, they increase it less than it would be optimal. That less-then-optimal increase of standardisation follows from the fact that when consumers choose a version to use, they only consider the effect of that decision on their own network benefits, but not on the network benefits of other consumers. It follows that under open source one should expect less than optimal standardisation, because varieties are chosen independently by distributors or consumers themselves without taking into account the negative externalities on other agents.

How does it compare with the willingness of a proprietary monopolist to standardise? Without network externalities, a monopolist can provide too little or too much diversity (Tirole, 1997, Ch.2). Too little, because he cannot appropriate the whole surplus that he creates by introducing new varieties. Or too much, if additional varieties allow the monopolist to raise the price without increasing the consumer surplus significantly. Compared to the choice of varieties by open source consumers or distributors, a monopolist can be expected to provide more standardisation, because it can appropriate network benefits. Moreover, a monopolist does not have a business-stealing incentive to introduce new varieties.

To conclude, it seems justified to say that open source programs are likely to be insufficiently standardised, that is there may be too many too diversified versions around. That may also have a negative influence on their probability of success: many users may choose a standardised proprietary program if they are afraid of too many incompatibilities between open source versions. Similarly, the producers of complementary goods prefer higher standardisation. However, this problem is likely to be minor in the case of programs where tastes are relatively homogeneous and network externalities strong.

6 Conclusions and welfare implications

This paper consists of a descriptive and an analytical part. The introduction and Sections 2 and 3 describe the phenomenon of open source, give an overview of its history, and list some common opinions about open source. Sections 4 and 5 proceed with an analysis of selected economic issues related to open source. In Section 4 I study the incentives of a proprietary incumbent to invest in quality, complementary goods and the consumer base, the choice of differentiation and compatibility with the competitor motivated by entry deterrence or accommodation of an open source entrant, as well as barriers to entry when the incumbent is open source. Section 6 focuses on the problem of insufficient standardisation of open source. Subsection 6.1. summarises the findings of sections 4 and 5, and subsection 6.2. discusses the welfare implications.

6.1 Summary of the findings

An introduction of an open source program into a market initially dominated by a proprietary program may have various effects, depending on the characteristics of the market and the entrant. If the features of the open source program are only attractive to a relatively small group of its developers, the incumbent may not find it worthwhile to try to discourage its entry. That will be most often the case if the users of the new program have a strong preference for those characteristics that are difficult to imitate by a commercial firm, like the accessibility and adjustability of the code. In this case, the open source program is likely to enter into a market niche.

What will be the impact of the open source on prices, investment in increasing quality, product differentiation and compatibility in this situation? The analysis conducted in section 4 does not allow making conclusive comparisons between the incumbent's incentives in the presence of proprietary and open source entrants. It is not clear whether it will invest more or less in quality, whether it will differentiate more, and whether it will allow more or less compatibility than with a proprietary competitor. One may, however, expect that because entry accommodation requires increased differentiation, the incumbent will pay relatively more attention to those aspects of quality in which it has an advantage over open source and which are important for the majority of users (for instance, user friendliness, availability of documentation and support). Furthermore, taking quality, differentiation and compatibility as given (and I did not find reasons to believe that they should be systematically higher or lower) an open source competitor decreases the price of the incumbent more than a proprietary one. Hence, one may expect lower prices after entry of open source (as compared to a proprietary entrant).

On the other hand, if the open source entrant has features that make it attractive for most users, it will have a strong negative impact on the profits of the incumbent, who then has more incentives to deter entry. Because of a more negative impact on prices, these incentives are likely to be stronger than when the entrant is proprietary entrant. If the attractive features of the entrant are not too difficult to imitate, then the incumbent will increase his investment in them order to discourage further development of the open source. Thus, entry deterrence will lead to an increased investment, especially in those features in which the entrant has an advantage and which are important for most users, and to avoiding any compatibility with the entrant. Eventually, the open source will either disappear, or it will become a niche product, used only by those who have a strong preference for open source.

Finally, if the open source program has features which are attractive for most users and which are difficult to imitate by a commercial firm, the incumbent may find deterrence too costly. That may be the case if most users value the access to the code highly, which is most likely if most of them are also programmers. Then, the incumbent will accommodate and either disappear, or become a niche product. If it stays in the market, it is not clear whether the incumbent will choose a higher or lower level of quality, differentiation or compatibility than in the presence of a proprietary entrant, but one may expect increased investment in those aspects of quality which are less well developed in open source, as well as lower prices. If the proprietary incumbent faces more (potential) competitors, the analysis becomes more complicated. The incumbent's incentives will then follow from the interaction with all entrants, both proprietary and open source.

If open source is an incumbent, its impact on entry, as compared to the impact of a proprietary one, is ambiguous. It is also ambiguous whether an open source incumbent encourages more open source or of proprietary entrant. The stronger the preference for inherent advantages of open source in the market, the less entry of proprietary firms may be expected.

As for the standardisation between different versions of open source, there seems to be a reason to think that insufficient standardisation will persist as a problem of open source software, although it will probably be minor in markets in which very strong network externalities force a high standardisation. In these markets open source software will be less at a disadvantage towards proprietary software.

6.2 Welfare implications

To analyse the impact of open source on welfare, one needs to compare the actual or expected market outcomes after its introduction with the situation before its introduction, and with the social optimum. Below I discuss some market outcomes: the impact of open source on competition, approximated with the impact on prices and entry, the influence on quality, differentiation and compatibility between competitors' programs, and standardisation between versions of the same program. When possible, the outcome is evaluated from the social welfare point of view.

(i) Prices

Open source distributes at zero price, which is its socially optimal price equal to the marginal cost. Other things (quality, differentiation, compatibility) being equal, the presence of open source should lower the price of other programs as well. Since I did not find reasons to believe that the choice of quality, differentiation or compatibility would be systematically biassed in one direction in the presence of open source, it seems reasonable to expect that the prices in general should be lower after it is introduced.

(ii) Entry

To evaluate the impact of open source on entry, one has to compare the incentives to enter when the incumbent is proprietary and open source. The results of the analysis conducted in section 4 are ambiguous. As regards proprietary entrants, open source on the one hand discourages entry because of its zero price, but on the other hand it may encourage it because it makes it easier to achieve compatibility. Similarly, it is not clear whether open source enters more easily when the incumbent is open source as well. On the one hand achieving compatibility is easier, on the other - the demand for open source is already satisfied by the incumbent. Hence, it is difficult to determine the impact of open source as an incumbent on entry.

The analysis has shown that open source discourages entry of proprietary entrants more when many users have a preference for open source. However, in these cases it may be socially optimal that the market is dominated by open source. Concluding, not enough reasons have been found to think that open source has either a positive, or a negative impact on welfare through the impact on entry.

The next three points discuss the impact of an open source entrant on welfare through the influence on the incentives of a proprietary incumbent to choose the level of quality, differentiation and compatibility. The framework of Fudenberg and Tirole, used in this paper, is not explicit about social welfare. Hence, I focus on specifying questions that should be answered when trying to determine the impact on social welfare. When possible, the answers to these questions are discussed as well.

(iii) Quality

To determine the impact on welfare through quality, one should know the incentives of a proprietary incumbent to invest in quality when faced with an open source entrant, as compared to a proprietary one. The results of the analysis ambiguous: the investment in quality can be higher in the face of a proprietary and an open source entrant, both in the case of entry deterrence and accommodation.

Furthermore, one needs to know whether the incumbent the incumbent will invest more when deterring open source or accommodating a proprietary incumbent, and vice versa. I have not been able to make this comparison, because of difficulties with comparing the incentives of open source and commercial developers. Knowing that, and the incentives to deter or accommodate both types of entrants, one should be able to say whether the incumbent will invest in quality more after the entry of open source. The resulting levels of investment should then be compared with the social optimum.

(iv) Differentiation

Similar questions should be answered when determining the impact on the level of differentiation. Also here the answers are ambiguous. When the incumbent accommodates entry, it can choose to differentiate more or less from an open source than a proprietary entrant. When deterring entry, it will try to imitate the advantages of the entrant, whether it is open source or proprietary. The incentives to deter or accommodate differ according to the characteristics of the entrant and the market: deterrence (and therefore low differentiation) is most likely if the entrant is attractive for most users and if the imitation of these attractive features is easy, and the least likely if only a small group has a preference for the entrant and if its advantages are difficult to imitate.

What is the socially optimal differentiation? In the Hotelling horizontal differentiation game discussed before, accommodation leads, whether the entrant is proprietary or open source, to maximal differentiation, which is too large from the social point of view. As for entry deterrence, it will lead to the imitation of the entrant, and possibly to too little differentiation.

(v) Compatibility

In the social optimum, all programs present in the market are perfectly compatible. When the incumbent accommodates entry, it is ambiguous whether it chooses more compatibility with a proprietary or an open source entrant. When it deters entry, it will never choose more compatibility with open source. This might suggest that in some cases an open source entrant may lead to less compatibility in the market.

Suppose now that the incumbent is open source. Are the incentives of its developers to choose quality, differentiation and compatibility larger or smaller than those of a commercial firm? Quality and differentiation are difficult to compare, because the incentives of open source developers are not known enough. As for compatibility, it is chosen by the entrant. Hence, if entrants wants more compatibility with an open source incumbent than a proprietary incumbent with entrants, open source incumbent will lead to a better, from the social point of view, degree of compatibility between different programs.

The last point refers to the analysis conducted in section 6 and summarises the effects of the choice of compatibility and standardisation between versions of the same program (before I discussed compatibility between programs made by different producers).

(vi) Standardisation

One may expect that open source programs will tend to be standardised less than proprietary programs, and that this standardisation will be insufficient.

Thus, the analysis suggests that entry of open source should have a positive effect on prices and on the incumbent's investment in those features of software which are valued by average users. If open source gets a large market share, it may have a negative impact on entry of proprietary firms and on standardisation. On the other hand, if open source is an incumbent, it is likely to be a signal that users have a strong preference for open source, and therefore that it is optimal there.

Other effects are ambiguous. This ambiguity follows largely from the fact that the incentives of open source and proprietary producers are of a different nature, and although the different sources of benefits can be identified, the strength of incentives is difficult to compare. This suggests that in order to draw more conclusions about the impact of open source, more research is needed: on the one hand empirical, into the incentives of open source developers, and on the other hand theoretical, into building a framework for comparing open source and proprietary producers.

Can something be said about whether the entry of Linux is good or bad for welfare that Linux is present in the market for the operating systems? It is not clear whether it should lead to more or less investment in Windows by Microsoft, although it is likely to lead to more investment in the features that are the most valuable for average users. If Linux already has these features, they are likely to be taken over by Windows - unless they are difficult to imitate for a proprietary firm, like accessibility of the code. Hence, if the developers of Linux have superior information about the tastes of average users, its entry may be welfare improving even if it does not get a high market share, because it may help (and motivates) the incumbent to provide for these tastes better.

If the attractive features of Linux are very costly to imitate - for instance, if it turns out that a proprietary firm cannot achieve the same level of reliability - then Linux may take over the market. Would it be better to have Linux as the incumbent? If most have an inherent valuation for open source, then it might be better. One may then expect little entry of proprietary entrants, but if the consumers prefer open source, that should not have a very negative impact on welfare. On the other hand, it may have a negative impact on the degree of standardisation in the market. All in all, I do not think that these findings give enough reasons for encouraging or discouraging the adoption of Linux by governments.

A similar answer can be given to the question whether it is good that Apache or Sendmail dominate in their respective markets, with the exception that standardisation does not seem to be a problem there. My guess would be that the fact that they became dominant is a signal of a strong preference of users for open source, and therefore open source is optimal there. Hence, I did not find a reason to intervene there.

Is there too little open source software in the markets? The paper did not really address this question. I did not find reasons to believe that there will be less entry of open source than proprietary programs, and I did not try to determine what the socially optimal entry of open source would be. There exist a large number of open source projects, which suggests that entering with an open source program is not very difficult.

Finally, I should mention that to make the analysis of the impact of open source more complete, one should also look at its impact of open source in complementary markets. Most business strategies related to open source adopted by software companies or identified in the literature (see e.g. Raymond, 2000d, or DiBona, Ockman and Stone (ed.), 1999) involve the production of complementary good or using open source as inputs. For instance, various distributors of Linux (Red Hat, Calder Systems Inc., SuSE, Corel Corp. and TurboLinux Inc.) make profits on putting together the pieces of the program, adding (sometimes proprietary) extensions, documentation, manuals and technical support. Another example is the publisher O'Reilly and Associates, who sells books and documentation for various open source programs. That might suggest that many commercial firms may have incentives to contribute to the development or in a different way support open source. An analysis of these incentives and their consequences goes beyond the scope of this paper, although it is likely to bring important insights into the impact of open source.

Appendix

This Appendix contains a list of some of the most popular open source programs and more detailed information on some of them. The descriptions of Apache, Perl and Sendmail are based mainly on the Lerner and Tirole (2000).

GNU/Linux operating system

GNU/Linux is a Unix-like operating system.[10] In 1991, a graduate student of Helsinki University, Linus Thorvalds, wrote an operating system kernel (core), based on Unix, for own use on his personal computer, and posted it on the Internet with a request to other programmers for help to develop it into an operating system. The work on the system attracted many developers from all over the world. Many solutions had already been available in the form of software developed by the Free Software Foundation of Richard Stallman. In 1994, the first version of the operating system, Linux 1.0, has been released. It is estimated that Linux has around 25% of the market for server operating systems, which places it on the second place after Windows NT with 38% (The Economist, June 17, 2000). According to John Hall, executive at Linux International, Linux has around 6% out of the worlds 450 mln of general purpose operating systems.

The work on Linux continues. It is estimated that around 1000 programmers contribute to the program. The project is developed in the modular way, typical for open source projects: the program is divided into relatively independent modules, which are managed by module leaders according to the "benevolent dictator" model. Everyone is free to contribute changes or comment on other's work, but the final decision to incorporate contributions into the module is made by the leader. At the head of the whole project stands Linus Thorvalds.

In computer markets, where any particular piece of software of hardware can only work as a part of the whole system, the success of an element like an operating system depends on whether availability of complementary components. It is also important that potential users expect that the complementary components will be available. Hence, announcements about "supporting" Linux by large hardware and software producers increase largely the chance for success. Hardware companies who decided to install Linux on (some of) their computers include for instance IBM (also on laptops), Dell, Compaq, Hewlett Packard. Some software companies, like Oracle, announced that they would provide technical support for Linux or write software that will run or it. Some companies have also decided to spend resources on developing Linux. However, since the results of the work are freely available, a free-riding problem is likely to arise. The establishment of the Open Source Development Lab, opened at the beginning of 2001,

[10] See also Raymond (2000a, 2000b) and Thorvalds, L., (1999).

financed co-operatively by among others Hewlett- Packard Co., IBM, Intel corp., and Nec Corp. may be an attempt to limit the free-riding problem.

Linux is also finding its way into the embedded software market, i.e. the market for software that is permanently installed on electronic equipment (for instance, Musical Stores Corporation in the US has decided to install it on its cash registers). In April 2001 Sony has announced that it will release a version of Linux to run on the Japanese version of its PlayStation2.

Linux is not only downloadable from a website, it is also distributed on CDs by a variety of companies. The prices vary from approaching the marginal cost of a copy on a CD and around 50 dollars charged by the largest distributor, Red Hat Linux (15-20 million users in 2000). The last price is clearly charged not for the raw information (code) but for the service of selecting, arranging and bundling the information to make a useful product for a user for whose cost of doing these services for him/herself would be larger. Moreover, many distributors guarantee support for users of their versions.

One of the serious problems facing Linux is a threat of fragmentation into many incompatible versions. The applications developers sometimes complain that they do not know for which distribution they should write their programs. To get around this problem, some of the main distributors, e.g. Red Hat, Calder Systems Inc., SuSE, Corel Corp. and TurboLinux Inc. are working on developing a common standard (see *ComputerWorld*, May 15, 2000).

In 2000, many Linux distributors suffered from significant losses and decrease of their stock exchange value, which forced them to scale down their activities. However, this was a general trend in the IT industry. It is therefore difficult to say whether the leaders of businesses based on open source overestimated their chances more than other IT entrepreneurs.

Apache

The development of Apache web server began in 1994. Most Internet servers in that time ran on Unix-based software written at the National Centre for Supercomputer Applications at the University of Illinois. Its code was freely distributed, and the development group co-operated actively with users. A mailing list was maintained such that enhancements to the code, often written by users, could be distributed. However, when the NCSA staff became sluggish in its responses to users' suggestions, seven pioneering developers established their own mailing list to collect and integrate patches.

As often with open source projects, the division of the program into modules, which allowed independent work by many programmers, was important. It was decided that the management of the project is done collectively by a small group of individuals. At present, Apache runs around 60% of all web servers and is the most popular web server operating software, which places it before Microsoft software (25%).

Perl

The Practical Extraction and Reporting Language was created and introduced on the Internet by Larry Wall of Burroughs in 1987. Wall faced the problem of many repetitive system administration tasks, and since none of the existing programming languages satisfied his needs, he decided to develop his own language. The idea was that the language would enable programmers to quickly undertake many administration tasks. After its introduction it has became popular as a language to develop small programs running with Apache web servers. The management of Perl is rotational: ten to twenty programmers manage different parts of the program in turn.

Sendmail

The Sendmail e-mail sending program has been developed by Eric Allman, who as a graduate student was responsible for many computer administration tasks at the University of California at Berkeley. One of the problems he faced was incompatibility of the main computer networks on campus: BerkNet, Arpanet, and the telephone network used to connect to other campuses. Each of them used a different communication protocol, and it was not possible to send e-mails from one network to another (so that having multiple accounts was often necessary. To deal with this, in 1981 Allman developed first Sendmail, which allowed sending e-mails between different networks. Thanks to that feature it became the standard way of routing messages on the Internet. However, after a while it split into many versions, often incompatible with one another. In 1993 Allman rewrote Sendmail, in a way that it made it incompatible with the older versions, but that also improved it sufficiently to drive the old versions out and become the dominant e-mail-sending program on the Internet (80% of e-mail traffic in 1998). In 1997, Allman established Sendmail, Inc., which sells services and enhancements to Sendmail, and continues to develop the program. Since 1998 it offers a proprietary version, SendmailPro, which runs one year ahead of the open source version.

Python

The development of Python programming language started in 1990 at the Centre for Mathematics and Computer Science (Centrum voor Wiskunde en Informatica) in Amsterdam, and was later continued by Corporation of National Research Initiatives in the United States. It can be used with various operating systems (Unix, Windows, DOS, and Mac). An organisation, Python Software Activity has been established to promote further development and co-ordination. The PSA also owns the copyrights on Python.

LaTeX
Text formatting program, used for writing mathematical formulas; it can be downloaded freely, but there exist commercial versions as well. It is widely used for scientific books and articles. It is maintained by LaTeX3 project, run by volunteers.

AbiWord
A text editor developed as a part of AbiSource project, which specialises in open source applications. AbiWord is available since 1999, also on CD. It can work on different platforms, among others Windows and Linux.

Other programs include:

- Free BSD, Unix based operating system developed at University of Berkeley, California).
- BIND (Berkeley Internet Name Daemon), program for managing Internet name domains on the Internet.
- Mozilla, an open source version of Netscape Communicator.
- GNOME (GNU Network object Model Environment), a project of Free Software Foundation to build a user-friendly desktop environment.
- KDE, K Desktop Environment for Linux.
- Junkbuster, server for filtering web content.
- PostgreSQL, a database of Great Bridge Norfolk; the company provides services and support Emacs, an e-mail program.
- K Office, office suite for K Desktop Environment including a spreadsheet and a text editor

References

Behlendorf, B., C. DiBona, S. Ockman and M. Stone (eds.), 1999, Open Source as a Business Strategy, Open sources: Voices from the Open Source Revolution, O'Reilly &Associates, Beijing.

Bradner, S., C. DiBona, S. Ockman and M. Stone, (eds.) 1999, " The Internet Engineering Task Force", Open sources: Voices from the Open Source Revolution, O'Reilly &Associates, Beijing.

Breedveld, P., A. Koldewe, R. Scharis en R. Westerhof, 1999, "De economische betekenis van Open Source Software in Nederland", International Data Corporation, (downloadable from EZ website, www.ez.nl).

ComputerWorld, May 15, 2000, "New Group Tackles Linux Compatibility".

Cottrell, T., 1998, "Microcomputer Platforms: Competition in software", in Duetsch, L., (ed.), Industry Studies, Prentice Hall, Armonk, N.Y.

Economides, N., "The Microsoft Antitrust Case", 2001, Journal of Industry, Competition and Trade: From Theory to Policy, vol.1, pp. 7-39.

Economides, N., "The Microsoft Antitrust Case: Rejoinder", 2001, Journal of Industry, Competition and Trade: From Theory to Policy, vol.1, pp. 71-79.

Farrell, J. and G. Saloner, 1986, "Standardisation and Variety", Economics Letters, vol. 20, pp. 71-74.

Fudenberg and J. Tirole, 1984, "The Fat Cat Effect, the Puppy Dog Ploy, and the Lean and Hungry Look", American Economic Review, vol. 74, pp. 361 - 366.

Hamerly, J., T. Paquin, S. Walton, C. DiBona, S. Ockman and M. Stone (eds.) 1999, "Freeing the Source: the Story of Mozilla", Open sources: Voices from the Open Source Revolution, O'Reilly &Associates, Beijing.

Hotelling, H., 1929, "Stability in Competition", Economic Journal, vol. 39, pp. 41-57.

Johnson, J., 2001, "Some Economics of Open Source Software", unpublished.

Katz, M. and C. Shapiro, 1985, "Network externalities, competition and compatibility", American Economic Review, vol. 75, pp. 424-440.

Liebowitz, S.J. and S.E. Margolis, 1999, “Winners, Losers and Microsoft: Competition and Antitrust in High Technology”, The Independent Institute, Oakland.

Lerner, J. and J. Tirole, 2000, "The Simple Economics of Open Source", NBER Working Paper 7600.

Navon, A., A. Shy and A.F. Thisse, 1995, “Product Differentiation in the Presence of Positive and Negative Network Effects”, CEPR Discussion Paper 1306.

O'Reilly, T., C. DiBona, S. Ockman and M. Stone (eds.) 1999, "Hardware, Software and Infoware", Open sources: Voices from the Open Source Revolution, O'Reilly &Associates, Beijing.

Raymond, E., 2000a (first published 1997), "A Brief History of Hackerdom", www.tuxedo.org/~esr/writings/cathedral-bazaar/hacker-history.

Raymond, E., 2000b (first published 1997), "The Cathedral and the Bazaar", www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral - bazaar.

Raymond, E., 2000c, (first published 1998), "Homesteading the Noosphere" www.tuxedo.org/~esr/writings/cathedral-bazaar/homesteading.

Raymond, E., 2000d (first published 1999), "The Magic Cauldron", www.tuxedo.org/~esr/writings/cathedral-bazaar/magic -cauldron.

Salop, 1979, "Monopolistic Competition with Outside Goods", Bell Journal of Economics, vol.10, pp. 141-156.

Shapiro, C. and H. Varian, 1999, “Information Rules: a Strategic Guide to the Network Economy”, Harvard Business School Press, Boston.

Thorvalds, L., C. DiBona, S. Ockman and M. Stone (eds.) 1999, "The Linux Edge", Open sources: Voices from the Open Source Revolution, O'Reilly &Associates, Beijing.

Tirole, J, 1997, “The theory of industrial organisation”, MIT Press.

The Economist, Survey: Software, April 12th, 2001.

Worock, G.A., F.R. Warren - Boulton, K.C. Baseman, D. Gabel and G. Weiman (eds.), 1998, "Exclusionary behavior in the market for operating system software: the case of Microsoft", Opening networks to competition, Kluwer Academic Publishers.

Vixie, P.,1999, C. DiBona, S. Ockman and M. Stone (eds.)"Software Engineering", Open sources: Voices from the Open Source Revolution, O'Reilly &Associates, Beijing.

Young, R., C. DiBona, S.Ockman and M. Stone (eds.), 1999, "Giving It Away: How Red Hat Software Stumbled Across a New Economic Model and Helped Improve an Industry", Open sources: Voices from the Open Source Revolution, O'Reilly &Associates, Beijing.