Abstract

The phenomenon of open-source software production still puzzles economists. How can a mode of production exist that is based on the contributions of many software programmers without providing any direct monetary incentives for them? And how can these programmers coordinate their actions without relying on either the guidance of the price mechanism or the direction through a superior? This paper provides answers to such questions by investigating the entire system of open source, taking into account various parameters of organizational design. It is argued that open source relies on the existence and careful synthesis of several organizational elements into a consistent system. In a comparative organizational analysis, the system of open source is compared to the organizational design of software production in more conventional firms. The advantages of a co-existence of open and closed source software and the effects on social welfare are highlighted.

JEL Classification: D02, H41, L13, L17, L86, P51
Keywords: open source, software, consistent systems, comparative analysis, organizational design

*Valuable comments by Augusto Ninni and participants of the XV AISSEC conference in Naples are appreciated. I enjoyed discussing the subject of open source with Justin P. Johnson, Samuel Lee, Nina Moisa and Jens Prüfer. This paper is adapted from Chapter 7 of Weiß, M. (2007): Efficient Organizational Design - Balancing Incentives and Power. Palgrave Macmillan, Basingstoke.

†Wilhelm Merton-Chair for International Banking and Finance; Johann Wolfgang Goethe-University, Frankfurt/Main; http://www.finance.uni-frankfurt.de/pers/weiss.html; Phone: ++49(0)69 798-28269; Mail: weiss@finance.uni-frankfurt.de or marco.weiss@eurogroup-consulting.de
1 Introduction

From Netscape to Mozilla In 1999 the market share of the pioneering Netscape Navigator browser fell within a year from around 60 per cent to only 10 per cent. The browser war between Netscape Communications and Microsoft thus ended with devastating defeat for the company that had helped to shape the internet from its very beginnings in 1993. Since that time a new range of communication protocols, such as the hypertext transfer protocol, allows for the easy retrieval of information from the World Wide Web. The use of these protocols allows displaying resources on the internet on a graphical user interface if a particular software, a web browser, is used.

Mosaic was the first popular web browser that offered these new possibilities. It was developed in 1992 at the National Center for Supercomputing Applications (NCSA) at the University of Illinois by a team led by Marc Andreesen. The NCSA licensed the Mosaic browser on generous terms for various computer platforms. Non-commercial use was allowed for free, and access to the source code, the underlying set of instructions for the software, was possible, although the license was held proprietarily by the NCSA.

Seeing the potential of the internet, Marc Andreesen founded Netscape Communications Corporation in April 1994 to exploit the possibilities that web browsers offered commercially. Their product Netscape Navigator was officially released at the end of 1994. It was an easy to use product that soon became the dominant web browser with at times a market share of nearly 70 per cent. The innovation of showing some resources of web sites while other resources were still downloaded allowed for a more rapid display of the content and allowed reading the text while waiting for the graphics. This simplification and the decision to give the browser away for free reinforced the adoption of the internet by numerous users. Despite broadly adhering to industry-wide standards, having the biggest share of the browser market allowed Netscape to influence the future direction of these evolving standards and gave it a superior position in the industry. Netscape maintained its technical leadership and used its dominant position to introduce technologies such as cookies, frames, and JavaScript in order to leverage this position. It thereby threatened to level the playing field between different operating systems.

This threat and the powerful position that Netscape had acquired in the internet made Microsoft, the company dominating the software industry with its operating system Windows, nervous. Fearing the dilution of its power, Microsoft introduced a web browser of its own in 1995, the Internet Explorer. Like the Netscape Navigator, the Internet Explorer was based on the source code of the original Mosaic browser. Microsoft has licensed this code from Spyglass, the company set up by the University of Illinois to commercialize the Mosaic web browser. While clearly inferior at the beginning, the Internet Explorer from version 4.0 on gained ground against Netscape’s Navigator. Decisions by Microsoft to bundle its Internet Explorer into its operating system Windows and to allow internet service providers (ISPs) and computer vendors to customize the Internet Explorer to their products clearly helped it in the battle with Netscape.

While the Internet Explorer was getting better, Netscape lost its technological lead. The decision to offer an e-mail client and a web site composer along with its Navigator in its application suite Netscape Communicator made the source code complex. As a consequence, the Navigator browser become buggy, prone to crashes, and could not handle the increasingly complex code of web sites that relied on JavaScript and tables. Finally, web developers started to use Microsoft’s proprietary extensions of the hypertext markup language (html) that favored the Internet Explorer and accelerated Netscape’s demise leading to a radical switch in the dominant system.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet Explorer</td>
<td>82.0%</td>
<td>71.0%</td>
<td>62.0%</td>
<td>56.0%</td>
</tr>
<tr>
<td>Gecko (Firefox, Mozilla)</td>
<td>14.0%</td>
<td>24.0%</td>
<td>32.0%</td>
<td>39.0%</td>
</tr>
<tr>
<td>Opera</td>
<td>2.0%</td>
<td>2.2%</td>
<td>2.0%</td>
<td>2.2%</td>
</tr>
<tr>
<td>KHTML (Konqueror, Safari)</td>
<td>1.1%</td>
<td>2.6%</td>
<td>4.2%</td>
<td>3.5%</td>
</tr>
</tbody>
</table>


Table 1: Market share of different web browsers

After defeat, Netscape announced on 22 January 1998 to reveal the source code of its Communicator under an open-source license in order to garner development effort from outside Netscape. These efforts culminated in the Mozilla project. Netscape Communications Corporation was finally acquired in 1999 by AOL, an internet service provider, that eventually discontinued the development in 2003.

Having undertaken such a dramatic step of opening the source code, some difficulties existed at the start. Building the Mozilla web browser from Navigator’s source code, for example, still required a proprietary library from a third party and did not work properly due to the huge and complex code base which proved difficult to disentangle. Most of the core developers decided therefore to scrap this legacy code and to rewrite important libraries and the rendering engine. A short time after that decision, on 7 December 1998, the new Gecko engine was released that was noticeably faster and smaller in terms of lines of code than its predecessor.

With AOL’s decision to close its browser division, the Mozilla Foundation took over the responsibility for the development of an alternative web browser to Microsoft’s Internet Explorer. Being fully independent from any commercial interests, the Mozilla Foundation can follow the significantly different production processes in open source. The integrated applications suite of Mozilla, which encompasses a browser, an e-mail client, a newsreader, and an html editor, was split into more manageable modules that still rely on common underlying technologies like the Gecko rendering engine but allow for a more reliable and secure program. The e-mail client has since been released as Thunderbird and the web browser on 9 November 2004 as Firefox. During the first 24 hours after the initial release, Firefox was downloaded more than a million times, and more than 50 million downloads occurred in the half year after its release. As Table 1 shows, the percentage of people switching from the Internet Explorer to a web browser based on the Gecko rendering engine developed and maintained by the Mozilla open-source community is on the rise.

Methods and objective This introductory example leads to many questions and puzzles: For example, where does this recent success of Firefox and Mozilla come from? How did the open-source project Mozilla succeed, where its closed-source predecessor Netscape failed? And, in an industry characterized by network effects and economies of scale and scope in production that often lead to a quasi-natural monopoly, with which organizational design could the position of the powerful incumbent be overcome without taking resort to heavy regulation? To answer these questions this article uses the framework of an efficient organizational design developed in Weiß.

\[1\] The name Mozilla was the codename in the development of the original Netscape browser. It is a contraction of Mosaic killer to clarify the intention in its development. The company mascot of Netscape, a Godzilla-like monster, was also of this name.

\[2\] See Raymond (2000a, p. 32).

\[3\] The rendering engine is a piece of software responsible for how content and format information is displayed by the web browser or another program. It can be embedded in the overall software program or can be a separate component as is the case for the Gecko engine on which several web browsers are based.
(2007). It is argued here that open-source organizations, designed correctly, can be a means to stimulate competition in an industry where network effects and economies of scale and scope exist, as shown by the rise of Mozilla and Firefox out of the ashes of Netscape. Where traditional firms, striving to make a profit, have failed or would not even dare to tread with the economies prevailing in the rugged business landscape so much against them, open-source communities can, nevertheless, constitute an organizational design that bypasses such economies.

Organizations are complex systems, and the way they are designed matters. Organizations can be decomposed into different modules such as their strategy, their boundaries, their internal structure and their governance. Within each of these four modules different organizational parameters can be configured distinctly to allow the organization to achieve its purpose in an efficient way. In the module of governance, for example, organizational parameters such as the relative importance of the different constituencies and the means by which power is allocated to them matters. Different configurations are usually possible for each parameter. In the case of the governance modes that convey power, this could be the assignment of ownership rights, the ease of exiting an organization, the possibilities for active participation through the use of voice or the provision of access to important resources.

The organizational parameters are often characterized by complementarities between them. Complementarity, or supermodularity, is a concept of systems theory. It is given mathematically if the cross-derivative in an objective function containing these parameters is positive. If complementarities exist, it is thus worthwhile to increase (or introduce) one parameter if the value of another parameter is also increased (or the feature introduced). In the presence of complementarities only distinct sets of configurations fit to each other. The efficiency of any organizational design is maximized if use of these complementary relationships between the elements and modules of an organization is made. In this case the organization is configured for consistency, and returns from supermodularity result.

The modules and elements in the design of organizations are thus held together by consistency just as a capstone in an arch supports the whole structure with only its weight as illustrated in Figure 1.

![Figure 1: Consistency as the capstone of organizational design](image)

Applying the thinking of systems theory to the organization of economic activities leads to a second insight. If the efficiency of a system is maximized through consistency of its elements, neither an incremental deviation in the value of these elements nor an adjustment of the value of only one element is likely to improve it: A consistent design represents a local optimum,

---

4A formal approach to systems theory is offered by Topkis (1998).

5See Weiß (2007, p. 88f.).
and unlike the ideal settings imagined in basic microeconomics no single peak in the business landscape exists. Instead the business landscape is quite rugged as sketched in Figure 2. Many peaks and troughs prevail that represent different business models characterized by a unique set of consistent configurations of its organizational elements and modules.

![Rugged business landscape](image)

Figure 2: Rugged business landscape

The objective of this paper is to describe two of these peaks in more detail, namely the traditional business model of closed source firms and the design of open source projects. The distinct configurations of important parameters and modules of organizational design are explored, and critical success factors that are necessary for open source to work successfully are identified. Complementarities between the different organizational modules are highlighted, and the consistency of each design is demonstrated. In a comparative organizational analysis the two systems are contrasted to each other and investigated for their relative efficiency to answer the question whether one design is superior to the other.

**Some related literature** The phenomenon of open source has received a lot of attention in recent years. Along with the number of open-source projects and programmers has increased the number of engineers, sociologists, and researchers from other academic disciplines who all wonder about the rise of open-source software. Economists are no exception. Different theories and methods, ranging from field studies based on interviews of programmers to case studies of specific projects to economic models, have been applied to deal with the phenomenon of open source.

One major strand of the literature deals with the motivation of individuals to take part in an open-source community. Besides intrinsic motivation, low costs for participation and the signalling of abilities are seen as motives for people to devote their time and effort to open-source communities. The argument of low costs for participation has gained prominence with the communication via the internet becoming ever easier and less costly. This view is often supplemented with the benefits, or the external rewards, that can be obtained. Lakhani and von Hippel (2003), for example, emphasize the advantages of a system in which the developers

---

6The variety of research into the topic of open source is best reflected in the collection by Feller, Fitzgerald, Hissam, and Lakhani (2005).

7Data on the motivation to work for open-source projects is provided in two surveys of open-source programmers by Hars and Ou (2002) and Hertel, Niedner, and Herrmann (2003).
of software are also its main users and in which users help other users by providing valuable feedback. Open-source programmers, thereby, profit both by learning on the production side and by obtaining a better product on the consumption side. Rapid feedback by the best in each specific field or project allows for rapid advancement along the learning curve by individual programmers and alleviates the free-rider problem.\textsuperscript{8} The argument of open source acting as a signalling device rests on the insight that not only the recognition by peers and other insiders in an open-source community is valuable. If a signal about the quality of particular programmers can credibly be acquired that discerns these people from less able programmers and if such a signal is understood by outsiders, higher economic rents can be obtained by these individuals in the future. Such able people are thus willing to expend time and effort on writing and revealing software code.\textsuperscript{3}

A second major strand of literature deals with the economics of open source and how promising business models, especially contrasted with that of traditional, closed-source firms, can look like. Various formal models exist that focus on the competition between different regimes of software production. Particular aspects of competition between open source and closed source are highlighted, but, being analytical, these models necessarily neglect the importance of other parameters in the organizational design of open source.\textsuperscript{10} Case studies, on the other hand, allow taking the interdependencies between such a larger number of parameters into consideration. While many case studies of individual projects can be found, the description of key configurations of open source common to many projects is rarer. Lerner and Tirole (2002) provide valuable thoughts on the economics behind a number of important elements in open source, but they lack a more systemic perspective.

Garzarelli (2003) comes closer to this task. He relates open source to the economics of organization and stresses the importance of knowledge in the production of software. Organizational forms like clubs or communities allow people in the software profession to absorb any knowledge embedded in the program and its source code easily. This helps to progress with developing the code and reinforces the value of the program. A spontaneous order within these communities and a certain degree of self-selection of programmers among different tasks are then the result. A second contribution that is closely related to the objective of this paper to describe the systemic features of open source is the well-known contribution of Raymond (2000a). He compares the functioning of the open-source model to a bazaar and contrasts this with the business model of traditional software firms. It is the purposes of this article to extend this line of argument and to explore the differences in organizational designs between the open-source and the traditional model of software production.

**Outline** Section 2 provides background information on the software industry and the development of software in general and the phenomenon and licensing of open source in particular. Sec-

---

\textsuperscript{8}This effect is modeled by Johnson (2002). He shows that under certain circumstances the free-riding problem in the provision of a public good like open-source software can be overcome.

\textsuperscript{9}See the argumentation in Lerner and Tirole (2002) and the models provided by Leppämäki and Mustonen (2003), Prüfer (2004), and Lee, Moisa, and Weiβ (2004).

\textsuperscript{10}Mustonen (2003), for example, provides a model on the competitive interplay between a closed-source monopolist and a competing program based in open source. Parameters such as occupational choices made by programmers, implementation costs of users, and the wage offered by the monopolist determine the equilibrium between open and closed source. See also Dalle and Jullien (2001) or van Wegberg and Berends (2000) for these kinds of models.

\textsuperscript{11}See, for example, the case studies of the Mozilla project by Reis and Fortes (2002), of the Linux project by Moon and Sproull (2000) and Dafermos (2001) or of the Apache project by Franke and von Hippel (2003) and Mockus, Fielding, and Herbsleb (2000).
tion 3 and Section 4 describe the traditional business model of keeping the source code proprietary and the new business model of the open-source system with their key parameters, respectively. Possibilities for commercialization, paramount with proprietary software, are described for the open-source business model as well. In a comparative organizational analysis, these two distinct approaches are contrasted with each other in Section 5. Remarks on how the co-existence of both an open-source and a closed-source system leads to a more efficient organizational design of the overall economy conclude this paper.

2 The software industry

2.1 Technology and economics of software production

Technology of software production Today’s large software programs can easily contain several million source lines of code (SLOC)\textsuperscript{12} and, as Brooks (1975, p. 18 - 19) has observed, the costs of communication among programmers eventually outweigh any benefits from taking an increasing number of programmers on to a single project. The necessity of having them all working on a controlled version of the program would increase complexity tremendously. How is the complexity inherent in a software program kept in check? The answer is through modularity in the design of a software program, as described by Baldwin and Clark (2000) and Langlois (2002).

Modularization to reduce complexity can be found everywhere. A prime example is the separation of hardware and software in information technology at the beginning of the 1980s. In the beginning of information technology, computers and machines were built for specific purposes, and every new task that had to be performed required the wiring of the computer to be changed. This was time-consuming and it paid to build machines for more general purposes and specialize them through particular software programs which allowed to change the purpose and possibilities of the computer more easily than through rewiring the entire hardware. The emergence of the software industry is the result of modularity in design, and the decomposition helps to reduce complexity. The use of standards in the form of Application Programming Interfaces (API) among operating system and application software is a second example for how software development is simplified: Standardized interfaces between different pieces of software allow for a modularization in software production. A third and important example is the rise of object-oriented programming in the late 1980s. In this programming paradigm, a particular functionality is packed into an object whose internal running is encapsulated and which works with other objects only over certain interfaces. Such objects represent previously codified knowledge that can easily be called upon when the need for it arises. Object-oriented programming thus allows for the reuse of existing code, thereby reducing the overall development time of a program.

The Netscape Navigator’s demise described at the beginning of this article is to some extent the result of sacrificing modularity. Ever more features were introduced in a short time to gain the upper hand in the browser war with Microsoft. Mingling the code, however, was no good idea, as it ever grew larger and impaired the clarity in the design. This proved to be a considerable problem when the source code was opened in 1998 and contributions from the open-source community were solicited. Most parts of the code were subsequently rewritten to make it consistent with the business model of open source.

Modularization is not only a feature of a software program but also of the process that leads

\textsuperscript{12}Wheeler (2002) reports that the operating system Redhat 7.1 contains more than 30 million physical source lines of code. Later versions of Linux or other operating systems have surely further increased that number.
to this product. Software production technology can be broken into six distinct processes. The requirements have to be analyzed, modules of the software have to be designed, the code has to be written or reused, the code has to be tested for its functionality and quality, the program has to be built with the integration of the different modules, and the program has to be optimized. Designing software programs and the technology to produce them for modularity therefore allows parallel developments which speed up the finishing of the project.

**Economics of software production** To better understand the nature of competition in the industry it is helpful to recall some of the economics prevailing in the software industry. Development and deployment of software is characterized by a cost structure skewed towards large upfront costs on the side of the developers and implementation costs on the side of the users. The first characteristic gives rise to economies of scale and scope in production. Developers have to invest significant amounts of money to design, code, test, and maintain a program before they receive any revenue from the software. Once the program is developed, variable costs are negligible. Putting the compiled version on a server connected to the internet is often enough to distribute the software. Albeit there have been some experiments with charging for software on a subscription basis, most software is still licensed for a particular version of the program with the necessity to renew the license for an upgrade version of the software.

The second characteristic, implementation costs on the side of users, makes switching from one program to another difficult. Users often have to learn how to operate a particular program. The co-specialization of their knowledge towards the software leads to an effect of lock-in, and switching to a competing product becomes more difficult: Another learning process would have to be incurred. Switching costs for users are, furthermore, increased if considerable numbers of other users with whom frequent interaction is necessary have also adopted a particular software. Modifying files created with a certain software often demands the use of that particular software by other users as well. The result are network effects in the use of software.

Both characteristics, economies of scale and scope in production and network effects in the usage of software, lead to the tendency for software programs to become standards. Such standardization has its merits and drawbacks. On the one hand, people using the same program can more easily work together on a document without bothering about converting it from one program to the other. On the other hand, such standardization can lead to a monopoly where one company dominates the market and charges a high price for a product of relatively low quality and further entrenches its position through a foreclosure strategy in related markets. The introductory example of the Netscape Navigator and the Internet Explorer with their, at different times, high market share has demonstrated the prevalence of standards in the software industry.

### 2.2 What is open-source software?

**Definition of open source** Different definitions of open source exist and the understanding what actually constitutes open source varies. Open-source activities are promoted by two institutions. The Open Source Initiative (OSI) is a "non-profit corporation formed to educate about and advocate for the benefits of open source and to build bridges among different constituencies in the open-source community." To qualify as open source, software has to meet certain requirements. The Open Source Definition maintained by the OSI demands among other things that the redistribution is free and has to include the source code. Modifications and derived works

---

13 See Feller and Fitzgerald (2002) for a more comprehensive introduction to open source and its history.
based on the original open-source software must also be allowed.\footnote{See http://www.opensource.org/docs/osd, accessed 25 April 2007, for all ten criteria.}

The Free Software Foundation (FSF) is the second major institution ”dedicated to promoting computer users’ rights to use, study, copy, modify, and redistribute computer programs”\footnote{http://www.fsf.org, accessed 25 April 2007.}. The FSF is to a certain extent competing with the Open Source Initiative and has set up its own definition of `free software’ in contrast to `open source’ software. Freedom to use and modify software programs as well as access to the source code are requirements for software to qualify as free. The actual differences between free and open software are, however, quite small\footnote{See http://www.gnu.org/philosophy/categories.html, accessed 25 April 2007.} and are more due to ideology and philosophy than substance. The Free Software Foundation is the main sponsor of the GNU/Linux operating system and its founder, Richard Stallman, has co-authored the licenses that most open-source programs use.

**The General Public License** There are currently 58 different licenses certified by the Open Source Initiative.\footnote{See http://www.opensource.org/licenses, accessed 25 April 2007.} The most commonly used license among the many projects listed by sourceforge.net, the world’s largest open-source software development web site, are the GNU General Public License (GPL) in 65 per cent of all projects and the GNU Lesser General Public License (LGPL) in 11 per cent of all projects. Both licenses are maintained by the Free Software Foundation. Other often used licenses include the BSD and MIT Licenses as well as the Mozilla Public License (MPL).

The intention of the GPL is ”to guarantee your freedom to share and change free software - to make sure the software is free for all its users.”\footnote{http://www.opensource.org/licenses/gpl-license.php, accessed 25 April 2007.} Free does not, however, refer to the price charged for the software but rather to the unrestricted access to the source code. The difference between the GPL and the LGPL is its viral nature: Every piece of software using any code released under the GPL must be distributed under these terms as well. The LGPL in contrast does not call for this necessity: A module or library released under this version of the license, which is simply called by another software program without being modified does not restrict the choice of license for this other program. It is used for objects such as common runtime libraries.\footnote{A more comprehensive discussion of the differences between commonly used open-source licenses can be found in Lerner and Tirole (2005). Confer also O’Mahony (2003) for more information on the legal issues and possibilities to protect work in the commons.}

The GPL provides a legal framework that credibly commits programmers using previously written code to publish and release the derived product under terms of open source again. Original contributors to the code are hence reassured that code they have once written and donated to the open source domain cannot be used in any commercial program for which they would have to pay a licence fee to use or modify it. The ’copyleft’ provided by the GNU General Public Licence or similar OSI-certified licenses is hence an important complementary element to retain the openness of the source code.

### 2.3 Modes of innovation

The pursuit of profit induces entrepreneurs or commercial firms to invest in innovative activities that can bring forth new products or a better process. Being the first to have developed a new product allows to reap economic rents from this monopoly position. If this powerful position is secured ex post for some time, it allows to recoup the ex ante costs of investing in innovative activities and thereby stimulates such investments. The safeguarding of such monopoly positions
can be secured, for example, through the use of a patent granted temporarily by a government. In the software industry, the licensing of a program only in its binary form without revealing the source code effectively limits the possibility to copy the product and erode the advantage that was generated by research and development activity, since a reverse engineering of the original source code from the binary version is next to impossible.

This private mode of innovation has, however, its drawbacks. Monopoly positions, be they based on a government-granted patent or the clever restriction of access to the knowledge behind a product, impose their costs on society. The monopolist can not only extract rents from the consumers, he also makes it difficult to advance from such an innovation further by improving on the idea behind the patent or the knowledge embodied in the source code. The mode of open, or collective, innovation avoids the welfare loss associated with such a monopoly resulting from private innovation.\textsuperscript{21} With open innovation, new insights are freely revealed to others, so that they can build upon this knowledge to the benefit of all. Open innovation, however, has also its drawbacks: It suffers from the lack of incentives. Individuals are often disinclined to invest into research when the associated costs are too high. Waiting for somebody else to innovate becomes attractive and underinvestment prevails. A combination of private and open innovation can therefore be a more efficient design.\textsuperscript{22}

3 The system of proprietary software

3.1 Organizational design of proprietary source firms

The organizational design of the traditional, proprietary software development model is deployed by commercial firms, such as Microsoft, Oracle or SAP, which operate along the lines described in conventional textbooks on economics and management. The pursuit of profit is high on the agenda of these firms, which keep access to the source code of their products restricted.

**Strategy** The strategy of software firms rests on safeguarding the economic rents generated through their intellectual property and to maximize that part of the rents which accrues to their founder, his fellow shareholders, and key programmers. The intellectual property is protected by closing the source code and is commercialized by licensing the software only in its binary form to users. Adding to the source code and licensing such upgrade versions translates into future profits and makes any competitive advantage of such companies sustainable.

The software industry as a relatively young industry is characterized through companies in which the founders usually still play prominent roles as the biggest shareholders or, additionally, as executive managers. Incorporating allows the entrepreneurs with an innovative idea to capitalize on their intellectual property. It allows them to ask shareholders providing capital and key programmers contributing their work to make co-specialized investments. Thereby it becomes possible to accelerate the development and to scale up the business. Being the first company in a market often characterized by a winner-takes-all competition allows the firm to seize upon the economies of scale and scope in their segment of the software industry and to benefit from network effects as more and more people start to use their program. This entrenches the firm in a powerful position and reinforces the possibility to extract economic rents to the advantage of its shareholders. The strategy of software companies, therefore, often rests on innovation and

\textsuperscript{21}The idea of open or collective innovation was first described by Olson (1967) in the context of the provision of public goods.

\textsuperscript{22}For the conditions that allow innovation to proceed in such a mixed mode, see the private-collective model by von Hippel and von Krogh (2003). See also von Hippel (2005) and Weiß (2007) and the examples therein.
the subsequent drive to establish the program as the standard software in that particular market segment.

**External boundaries and internal structure**  
The decision where to set the boundaries of the firm complements the strategy just described. One important configuration is the choice where to locate the company. The need to tightly control the access to the source code increases the payoffs from centralizing the programmers in one location. A close geographical proximity of programmers in few development centers usually characterizes firms operating along the lines of the traditional business model.

There exist further possibilities to configure the organizational design of commercial software companies. Strategic alliances, for example, are a promising form to enhance the established base so important in any standard war. Firms in software development of all sizes, furthermore, cluster together in particular regions like Silicon Valley in the USA or around Bangalore in India. This facilitates the transfer of talented programmers across the boundaries of the firm without imposing on them the necessity to move places. Boundaries are hence relatively permeable. What is, however, held tightly within the confines of the firm is the source code as the basis for future profits.

The internal structure of commercial software companies is characterized by internal boundaries complementing the external ones. Erecting barriers internally is done for various purposes. A hierarchy allows to use organizational parameters like screening of programmers for more sophisticated tasks or promotions as incentive devices. Clear assignments of tasks to programmers are possible with a hierarchy, and the information generated by continuous screening can be used facilitating coordination and centralized planning. With internal boundaries the power of different stakeholders, e.g. of lower-rank employees, can be restricted. Information and knowledge, especially that embedded in the source code, can be classified. Restricting the access to the code in its human-readable form thus complements the strategic choice to exploit the innovation commercially.

**Governance**  
Software firms can take on many organizational forms. Sole proprietorships and professional partnerships exist alongside huge and valuable corporations like Microsoft or SAP. As indicated before, the stakeholding constituencies include the founding entrepreneurs, the shareholders, senior managers, and key programmers. Customers often have to undertake co-specialized specific investments to deploy the software to their needs or to learn how to operate it. As in any corporation, ownership rights are assigned to shareholders, although through their usually significant shareholdings, or their moral authority, the founders often still retain control. In the software industry, employees often hold some amount of the share capital, especially if the use of stock options is used as an incentive device in the internal structure of such an organizational design. Customers, however, usually lack special protection of their stake through some governance mechanism. Reliance on the exit mechanism provided by outside markets is rather crude. Too much value of co-specialized investment is often lost.

### 3.2 Consistency and shortcomings

**The cathedral**  
The organizational design of proprietary software is a consistent system in which the important elements reinforce each other. No small deviation in the configuration of a single element can increase efficiency. The systemic nature of the design of commercial software companies is acknowledged by Raymond (2000a). In his essay 'The Cathedral and the Bazaar' he
likens the development of a complex program in a commercial software company to the building of a medieval cathedral: A well-known architect makes a blueprint of the building starting with the basic structures and ending up with the very details and ornaments of the cathedral. Sticking to that blueprint, the actual work is done by people who are experts in their narrow domain and who have the necessary specialized equipment to perform that particular task. The architect supervises and monitors the progress and adjusts the grand blueprint whenever the need arises, although careful planning at the very beginning should make this unnecessary. It is more often the individual worker who has to find a solution to match the overall design requirements. The cathedral model is characterized by a strong role for the central planner who has the knowledge to generate the blueprint. Building or production takes its time since careful planning, coordination, and supervision are necessary. The architect or leader of the project is the scarce resource and often the bottleneck in the process. Parallel work is nearly ruled out.

The organizational design of commercial software companies follows the building of cathedrals according to Raymond (2000a). Important configurations in the system of proprietary software include a strong role for the central authority and the reliance on leadership and formal hierarchy. The strong formal or moral authority of the founder reflects this. Such a strong role of leadership increases the returns from following strategic demands to become the dominant program in the particular market segment. Careful management of the boundaries, external as well as internal, and tight control of access to the source code fits with the configuration of the other parameters since it increases the amount of rents that can be appropriated. This spills positively over into the incentives for the various constituencies to invest specifically into the development of new and the maintenance of existing software programs. The relatively protected position, however, facilitates the appropriation of economic rents and can lead to complacency of the entrenched quasi-monopolist. Innovative activity is impaired as a result, and it often takes a long time for the final product to reach its users. This makes the system vulnerable to sudden changes which might render the business model pursued inappropriate in the future.

**Shortcomings of the traditional business model** Two shortcomings of the proprietary software system can be imagined. One concerns the appropriateness of the system in light of external changes, the other the efficiency of the wider economy of which commercial software firms are a part.

Changes in the rugged landscape of competition occur, and especially the technological progress in information processing and information transfer has an impact on the appropriateness of the traditional business model of commercial software companies. Is the system of proprietary software coding still appropriate for an environment characterized by the internet with its possibilities to copy software easily and implement it fast? The internet also fundamentally changed the possibilities for communication of people dispersed around the globe. New forms of cooperation have become possible. Automation in some parts of the software development chain, such as automatically building and testing the program overnight, further reduces the need for close geographical proximity of the different programmers.

The second shortcoming of the system of proprietary software is that it often fails to achieve efficiency in the larger organizational design of the overall economy. Network effects and standardization lead over time to monopolies. This reduces choice for customers and can make the incumbent quasi-monopolist reluctant to change and complacent if problems occur. Customiza-

---

23Note that other industries, such as e.g. the music business, are effected as well by the internet and its possibilities for sharing digital content over peer-to-peer networks among strangers. Business models resting on the restriction of access to knowledge-based assets or copyrighted materials seem to be threatened.
tion, for example, was not possible with the Windows operating system, and security holes caused by bugs in the programming are exploited by malicious hackers leading to serious problems for many users and businesses. The strategy of bundling together different services, albeit it may enhance the functioning of the components and their interplay, at the same time erects barriers to entry and further entrenches the position of the monopolist. Regulation is in many industries the solution to counter the power of such a monopolist. In the software industry, some kind of ad hoc regulation through legal actions can be observed. Different lawsuits brought against Microsoft and various fines for uncompetitive behavior levied upon it by American judges and the European Commission are the result. This is, however, only a crude way to counter ex post the power of any monopolist. The sum of all economic rents, i.e. including those that could be enjoyed by consumers or those lost due to the introduction of regulation, is not maximized. This organizational design does not maximize overall welfare.

4 The system of open-source software

4.1 Organizational design of open-source projects

Open-source projects release software under a license that requires the revelation of the source code. This change in the configuration of one important parameter, from keeping the source code proprietarily to publishing it openly, must lead to changes in the configuration of other elements as well in order to keep the organizational design consistent.

Strategy Open-source projects follow a fundamentally different approach to coding software than companies that keep the source code of their software proprietary. Rather than restricting access to the source code and to prohibit any modifications as a violation of copyright protection, a community of programmers actively encourages each other to build on ideas and code generated by fellow programmers. The vision behind many open-source projects is to provide an easy solution to common problems without the necessity to 'reinvent the wheel' every time again. Reciprocity within the community is an important pillar on which the strategy of such projects rest: The expectation to give and receive help from other members of the open-source community, if needed, is deeply embedded within the community.

Although well-known open-source programs such as Linux, or Apache command several hundred or even thousand programmers working on it, the number of developers in most open-source projects is relatively small. Often only one programmer develops and maintains the software project as Krishnamurthy (2002) reports. He notes, however, that with increasing maturity a selection among open-source projects towards successful ones occurs, and the number of people working on these projects increases. This induces individual programmers to actively market their product in the community and to create a scaled-down version that, nevertheless, attracts other developers early on. Frequent releases complement this strategy.

Once a project has succeeded in attracting more programmers, the success of the whole project is reinforced. Having many developers from various backgrounds using various hardware offers a superb testing environment to find flaws in the code and correct the program for these bugs. Different versions of the same program, distinguished, for example, in the case of Linux by an even number for stable versions and odd numbers for developments, often exist, so that users can self-select which one to use depending on their sophistication and inclination to co-develop the program.

Without relying on large amounts of money to write many lines of code for a complex program,
timeliness, functionality, and simplicity are important from the very beginning. Modularity in the design of the program helps to achieve this and allows for parallel development, thereby positively reinforcing the speed with which the product is finished. Modularity also decreases the complexity and simplifies the management of the project.

**Boundaries**  The boundaries between different open-source projects, and often also between open-source projects and commercial firms, are blurred to a degree of almost non-existence. Open boundaries complement the strategic importance of an open source code. Many programmers work on several projects at the same time or shift from one project to another with ease. Programmers employed by commercial companies, such as IBM or Nokia, actively contribute to open-source projects as part of their job. Open-source projects are therefore not clearly confined, and any existing boundaries are highly permeable. Different projects easily combine into one, cooperate extensively or split into their submodules that thus become viable open-source projects on their own. The latter is best illustrated with the case of the Firefox browser and the Thunderbird e-mail client spun off from the Mozilla project. Forking of projects, where the same software program develops into two or more distinct and competing directions, is also possible. Boundaries are hence often negligible and, applying the ideas of systems theory, would even be counterproductive in a consistent design of open source. The openness of the source code leaves nothing to protect from the outside. Erecting artificial barriers that make the crossing of resources across the boundary difficult would also run against the strategy of helping others and sharing information freely.

**Internal structure**  Many configurations in the internal structure support the strategy of open-source software development. A distinctly configured web of organizational elements characterizes the distribution of information and power of developers and users, provides incentives, and helps to coordinate dispersed investment decisions.

Open source builds upon superior availability of information. Knowledge resides in the source code itself and is accessible to everybody. A culture of actively sharing knowledge prevails, and transparency is high. Information and knowledge are generally used in an incremental way and, apart from the initial version of the software put into open source, small improvements in the software program prevail. This complements the use of frequent releases of the program for strategic reasons. The use of knowledge is not restricted by necessities to handle it sequentially since a modular production technology prevails. Disposing of rigid internal structures frees programmers to pursue working on that task which is most beneficial in their opinion. Simultaneous and parallel work becomes thus the norm in open source. This accelerates the speed of development. The design of open source is, furthermore, enhanced by the deployment of a tournament style in the development process. An increase in the value of parallel work due to complementarities in design increases also the benefits from relying on modularization, thereby positively reinforcing the other elements in the system of open source.

The management of complexity is simplified through other means as well. Enhanced possibilities for communication over the internet using newsgroups or specialized web sites such as SourceForge or freshmeat, dedicated project management tools like the Concurrent Versions System (CVS) or tools such as Bugzilla, which allows tracking bugs and managing their removal from the

---

24 Not everything, however, is open and transparent in the open-source community. The Mozilla Foundation, for example, restricts access to critical security issues to deter the exploitation of such vulnerabilities until a patch is available for the bug.

25 The Concurrent Versions System (CVS) is a source code collaboration tool. It allows to retrieve previous versions of a file by storing the differences between various versions. CVS also helps to manage the code of a
code, have all allowed open-source communities to handle the complexity inherent in software production more easily. It is, along with modularization, especially these innovations in coping with complexity that have led to the prominence of open source. Note that many open-source projects were explicitly initiated to support the wider community through the development of such utility programs. Any outsourcing of these ancillary modules into separate projects helps to keep the focus in the original project and is consistent with the blurredness of the boundaries. Modularization allows reusing the code easily, speeding up the development process, and decreasing the managerial complexity which, again, positively feeds back on the strategic importance of these configurations.

Power and incentives in the organizational design of open source are balanced in a congenial way and carefully adjusted to the information structure just described. The main pillar for assigning power and providing incentives is the peer review process. In this process senior programmers or the founder of the project who have an in-depth knowledge of a particular module and its interplay with other modules of the program evaluate software code by other programmers. They also manage the different tasks that have to be undertaken in the project by steering other people into making the most effective contributions given their particular abilities.

Becoming a peer and being able to make such evaluations requires an intensive knowledge of the code, which is usually gained through many contributions from their side. Power in open source is therefore acquired as a result of project-specific investments. The more code a particular programmer writes for a program, the higher his status in the peer review process. Advancing along a career path from the finding of bugs, the proposing of fixes or new code towards becoming a peer or finally a module owner provides incentives as well.

The peer review process is also consistent with some kind of hierarchy in the design of open source. A pyramid-like hierarchy among contributors seems to emerge naturally. Unlike hierarchies in traditional companies, however, the one in open source is a true meritocracy. The best programmers in terms of quantity and quality gain the reputation and the trust of their peers to direct the community and its future efforts. The system is self-organizing, and a selection of jobs by individuals according to their perception of existing capabilities and available tasks is the norm, strengthening the value of their local knowledge and circumventing the problem of quality uncertainty resulting from private information. Although most of the incentives are set in the form of peer recognition or a deferred payoff once the capabilities have become public knowledge, use of monetary incentives is made to a certain extent as well, especially for programmers employed by commercial companies operating in the wider ecosystem of open source.

**Governance** In the organizational design of open source, shareholders do not exist. Big financial contributions are not necessary in the organizational design of open source. They would arguably be even counterproductive, making the whole system inconsistent and hence less efficient. Software production is a human capital intensive industry, and financial capital is mainly needed for the purpose of speeding up the development process to reach critical mass for strategic reasons, as described above. Open source substitutes manpower for money. Employing a much larger workforce and using a technology resting on modularity that makes parallel work on a grand scale feasible allows open source to catch up with commercial companies and even take the lead in development of new products, as is the case with the Firefox browser described in the introduction. The human capital embedded in software programs is donated to the community project when several people are working on the project at the same time.

---

26See in this context Iannacci (2003) for a more detailed description of the Linux managing model. Confer also Osterloh, Rota, and Kuster (2002) for a description how the peer review process reinforces rapid production.
through small investments of time and effort of many different programmers. Such donations are possible since the roles of developer and user are combined. Economic rents are therefore consumed by those who create them in the first place.

The organizational form of a community or loose network is best suited to the demands in the configuration of other elements. Unlike other forms that restrict participation in the organization, open-source communities allow unrestricted access to their resources. Contact with the peers as the 'senior management' of the project is possible and welcome whenever the strategic direction of the project is taken into account.

Besides the governance mode of access, the possibility of exit is widely prevalent for everybody involved in the community. At the extreme, developers can 'fork' an open-source development project, effectively leaving a particular project and setting up a complementing, or sometimes competing, project. This allows them to add any critical components they need. Such a triggering of the exit option offers greater flexibility, and any user or developer can subsequently modify the program for his needs. The exit option coexists alongside the voice mode of governance. Those constituencies who want to acquire influence can do so easily and become a peer. All that is necessary are specific investments into the project, thereby further strengthening the crucial relationship between incentives and power as described above. Such behavior is immediately evident in the observation of the way that corporations in related industries, such as hardware vendors like IBM or software distributors like Redhat or Novell, endorse particular open-source projects and support these through various means, either financially or by actively developing code for the project with their employees. As argued below, such profit-oriented companies play a crucial part in the wider ecosystem of open-source communities.

4.2 Consistency and the open-source ecosystem

The bazaar The open-source system resembles a bazaar as Raymond (2000a) explains in his metaphor of a cathedral and a bazaar for the two different systems of software production. On a bazaar, stands are built wherever there is enough room to set up shop by whoever wants to. There is no central authority that permits to do so or directs individual merchants. Everybody is allowed to do whatever task he thinks is in his best interest. Network externalities are important. Such externalities make the individual investments worth more if other people also invest in setting up their stand at the same place and at the same time to attract the greatest number of customers. A well-established bazaar thus attracts other merchants, thereby reinforcing its success. A centrally planned coordination, however, is not necessary. A bazaar organizes itself whenever the purpose and available wares of the bazaar are common knowledge. From a dynamic perspective, it is worthwhile to note that a bazaar is dissolved in the evening to be set up from scratch the next morning. This allows for rapid changes in the meantime, unlike a building cast in stone like a cathedral which is erected for centuries rather than a few hours a day. Albeit minor or major restorations occur in cathedrals, there exists no possibility to change the basic architecture. A bazaar, on the other hand, is not built to last. It is intended to be deconstructed quickly. As such it is an efficient design if changes in the rugged business landscape are frequent and unexpected.

The business model of open-source projects resembles a bazaar according to Raymond (2000a). In open source, use of specific knowledge is made by explicitly giving people who hold this kind of information the power to decide. An organizational design automatically evolves over time and

---

27See Franck and Jungwirth (2003) on the importance of donation of software code in the governance of open source.
changes frequently subject to new local information. In such a system, leadership for coordination of dispersed decisions is necessary. If exercised for that purpose, it leads to successful open-source projects. Leadership and exercise of power for the main purpose of appropriation of economic rents, however, contradicts the configuration of other parameters in this design: The incremental nature of specific investments and the good exit opportunities in open source prohibit such a malign use of power. No merchant expropriated of the economic rent from his specific investments into a stand will return to the bazaar the next morning. 28

Nobody exercises ownership of the open-source software program in the sense of excluding others from the right to use it under the General Public License. Revenues from transferring or licensing this right, as is the case with commercial software, prove elusive. Economic rents need to be generated through small and incremental investments and have to be appropriated through the use of the final product by its developers who cooperate with each other. Through careful configurations of organizational parameters a system emerges in which incentives and power are both balanced, thus leading to an efficient design. The interdependence between different organizational parameters in the system of open source is illustrated in Figure 3.

The open-source system constitutes a consistent organizational design. Any change in the configuration of a single element does not increase efficiency. A caveat has to be mentioned, however, that might mitigate the applicability of open source: Getting the exact configuration of the organizational parameters right is a prerequisite for the functioning of the system. The design of open source, therefore, represents a peak with a steep precipice in the rugged landscape of competition. Even small deviations or a balance of power skewed towards a particular constituency can easily destroy the system. The system of open source is hence less robust than the system in which private innovation is a key design parameter.

Ecosystem of open source The organizational design of open-source projects described in an idealized way above does not exist in a vacuum. It is supplemented by a wider ecosystem consisting of various other institutions such as universities or commercial companies exploiting the

---

28See also Demil and Lecocq (2006) on bazaar governance.
products created in open source to make a profit from them. Their activities reinforce open-source projects by channeling additional resources such as necessary hardware, sponsoring of conferences or even employees dedicated to the ongoing development of particular projects to the community. Numerous business models exist, different strategies are pursued, and many propositions to create additional value by supplementing open source through commercial companies can be discerned. A short and non-comprehensive classification follows. For additional examples confer Raymond (2000b) or Schiff (2002, table 1).

- One of the most common business models in the ecosystem of open source is the provision of supplementary services. Distributors of open-source software, such as Redhat for the Linux operating system, bundle different software programs or modules such as the Linux kernel, the Gnome and KDE desktop environments, the OpenOffice suite, the Firefox web browser, and much more into different distributions in which the individual components work seamlessly together. While the distribution is still free for download from its web site, Redhat charges for ancillary services like complex installations and system administration, for the provision of continuous updates, for manuals and the training of users or for consulting services related to its distribution.

- Not only services supplement open-source software and the resulting system, products can do so likewise. The most prominent product to supplement software is traditionally the hardware called to life by the software. Established companies such as IBM enter the open-source domain to profit from complementarities between the two products. IBM is arguably the single most potent contributor to the community with a budget of several billion US-dollars related to open-source projects and a release of 500 patents into open source in January 2005.  

- Another example of how the deployment of an organizational design based on open source benefits the wider ecosystem can be found in the area of standard-setting. Agreeing on an open, industry-wide standard is an act of collective innovation. To accommodate the needs of all industry participants, it is necessary to keep the profit motive subdued. This can be achieved through the reliance on an organizational design following the layout of open source as described above. A blueprint for a business model along these lines of thought, albeit no open-source project in the narrow sense, is provided by Symbian. Symbian is a consortium of the leading mobile handset makers such as Nokia, Sony Ericsson, Siemens and others. By working together on a common operating system that makes the devices of different manufacturers compatible with each other, a sensible business proposition is occupied and value is generated. Users of mobile phones and the software running on it are also encouraged to participate in the development of the software, and more open-source projects evolve around Symbian.

- Many, if not most, open-source projects originate in academia. Universities are engaged in open source since it provides an ideal testing ground for training their students and supports many research projects. The subdued incentives in universities to commercialize successful open-source projects in an institutionalized manner complement the key characteristic of a free and open source code. Universities offer an ideal breeding ground to develop complementary products or services that can be exploited by individuals. The "ready access to venture capital"  that the proponents of open source have not only positively feeds back

29 Confer IBM (2005, p. 16). See also Henkel (2004) on the nexus of different firms as either users or sellers of a complement.

on their motivation but also allows them to scale up their open-source product through a business model along the lines described here. As the wider ecosystem of open source thus grows, so does the success of the organizational design of open source.

5 Comparative organizational analysis

In this section the configuration of organizational parameters in the two designs are contrasted to each other in a first step to determine whether one design is superior to the other. The interplay between the two systems and the effect on the efficiency of the larger design that constitutes the overall economy is discussed in a second step.

5.1 Comparison of the organizational designs

Synopsis of the two designs  Table 2 contrasts the differences in the configuration of important organizational design parameters for the two systems of software programming.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Proprietary source</th>
<th>Open source</th>
</tr>
</thead>
<tbody>
<tr>
<td>purpose</td>
<td>create a proprietary program as standard</td>
<td>create an ecosystem of compatible programs relying on open standards</td>
</tr>
<tr>
<td>product</td>
<td>emphasis on differentiation and innovation</td>
<td>emphasis on low-cost alternative</td>
</tr>
<tr>
<td>activities</td>
<td>integration of different pieces to a single complex program</td>
<td>reliance on modularity in design</td>
</tr>
<tr>
<td>duration</td>
<td>going concern, designed to last</td>
<td>designed for specific project</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Boundaries</th>
<th>proprietary source</th>
<th>Open source</th>
</tr>
</thead>
<tbody>
<tr>
<td>setting of boundaries</td>
<td>exclusiveness of resources</td>
<td>programmers working on several projects simultaneously</td>
</tr>
<tr>
<td>permeability</td>
<td>boundaries are difficult to cross</td>
<td>blurred and open, sometimes no perceivable boundaries exist</td>
</tr>
<tr>
<td>interactions across boundaries</td>
<td>alliances to create software standards</td>
<td>whole ecosystem of symbiotic relationships</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Internal structure</th>
<th>proprietary source</th>
<th>Open source</th>
</tr>
</thead>
<tbody>
<tr>
<td>use of information</td>
<td>screening of resources and direction</td>
<td>self-organization of resources through self-selection</td>
</tr>
<tr>
<td>power distribution</td>
<td>power assigned through departments and hierarchy</td>
<td>strong role of leadership, power contingent on specific investments</td>
</tr>
<tr>
<td>provision of incentives</td>
<td>stock options as incentive device</td>
<td>internal promotion and career concerns</td>
</tr>
<tr>
<td>coordination</td>
<td>ex ante planning and task assignment</td>
<td>CVS with approval of leadership and rapid releases</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Governance</th>
<th>proprietary source</th>
<th>Open source</th>
</tr>
</thead>
<tbody>
<tr>
<td>stakeholders</td>
<td>(founding) entrepreneur, employees, shareholders</td>
<td>(founding) programmers, users, distributors</td>
</tr>
<tr>
<td>mechanisms</td>
<td>ownership of shareholders, employee share ownership plans</td>
<td>access to peers, easy exit, acquisition of voice possible</td>
</tr>
<tr>
<td>organizational forms</td>
<td>corporations, partnerships, sole proprietorship</td>
<td>projects, communities, networks</td>
</tr>
</tbody>
</table>

Table 2: Synopsis of the designs of open and proprietary source

The different purposes underlying the two systems are reflected in the configuration of the parameters of strategy. A company operating along the lines of a business model resting on proprietary source code is keen to restrict competition through differentiation of and complexity in its product in order to maximize the economic rents for its shareholders. Having invested to entrench the own position, a dissolution of the design does not occur unless the venture is grossly unsuccessful. The design of open source, on the other hand, rests on compatibility of
different programs. Open standards and small modules hence prevail, and the resources employed in a specific project separate again once the project has achieved its usually narrow purpose. Dissolving an open-source project is simplified by highly permeable boundaries, and leaving is easy for programmers who often work on several projects simultaneously. Moving between different projects advances their career and status in the open-source community and acts additionally as an incentive device. In the design of proprietary software, boundaries are more restrictive to keep the source code secret and to protect the economic rents for the shareholders of the company arising from the strategically entrenched positions.

In the internal structure of open-source projects, power and incentives are aligned almost automatically. Those constituencies investing specifically into the success of the overall project generically acquire the power to safeguard the economic rents resulting from their investments. Power in proprietary software companies, on the other hand, is derived through the configuration of the various instruments of internal structure such as a superior position in the corporate hierarchy that facilitates coordination. Governance parameters such as the choice of the corporate form or the assignment of ownership rights to shareholders skew the distribution of power further. Obscuring the contribution of individual programmers to the software makes them less visible to the outside and lowers their exit opportunity. In sharp contrast to that, the visibility of individual programmers is enhanced in open source as argued by Lee, Moisà, and Weiß (2004) and is a key characteristic necessary for the functioning of the open-source system. Allowing developers and users to use the voice mechanism balances the power distribution in the communities and aligns it carefully with the necessary incentives for specific investments. Positions of power thus remain contestable as time goes by.

**Which design is more efficient?** The two idealized organizational designs described above differ in many configurations. Although both constitute consistent systems, one of them might be superior to the other forming a global optimum in the rugged landscape of competition. To answer the question which of the two organizational designs is relatively more efficient necessitates to define the dimensions along which efficiency can and should be measured. In the case of the production of software, suitable dimensions besides the costs of production are, for example, the quality of the software, the speed of development or the availability of solutions to frequent or less frequent problems. All factors influence the amount of economic rents generated by the organizational design. Additional factors can, of course, be imagined as well, but success is difficult to measure in most dimensions. Take the quality of a software program as an example. Not only the performance in terms of the execution of a given task is important, the usability also matters. The necessity to train users how to operate a particular program or the requirement to make the program compatible with other software through complex interfaces adds to the total cost of ownership, a concept commonly applied to measure the efficiency of a software program in financial terms. Economic rents, especially those accruing to users, are reduced in these cases. It is proprietary software that is often differentiated for strategic reasons, so that the amount of rents for particular constituencies of the organization is maximized disregarding economic rents for less powerful constituencies. This impairs the efficiency relative to a design in which power is more contingent on undertaking value-creating specific investments.

Another issue influencing the quality of the software is its security. In the case of web browsers and operating systems, this becomes obvious whenever another flaw in commonly used programs such as Microsoft’s Windows or Internet Explorer is exploited by malicious hackers. It is difficult to discern whether such security problems are a result of the production processes deployed in the traditional software model or whether they result from the determination to have the greatest
impact, in which case any malicious code is targeted against the prevailing software standard, i.e. often proprietary products by companies such as Microsoft. Whatever the answer to this question, the proponents of open source are probably correct in claiming that their products are at least better in terms of a quick removal of any flaws that are detected. Many programmers from diverse backgrounds yearn for high quality code that enhances their status and visibility in the community. Proponents of open source accuse proprietary software firms to achieve 'security through obscurity', implying that closed-source products remain secure due to the unavailability of the source code but only as long as nobody spots a weak point in it by chance. The same argument of having the larger potential workforce slightly favors open source along the second and third dimension mentioned above, the speed with which software can be developed and the availability of solutions to common problems.

Open source also beats proprietary software in terms of costs of licensing. The particular terms of the license used create a distinct distribution of power between producer and consumer in the product market. Software that is available for free allows its users to enjoy an economic rent from its use. For these reasons, i.e. due to the prevalence of the profit motive which leads to rent-seeking behavior, the organizational design of commercial software companies does usually not maximize the overall welfare. Only that share of economic rents that accrues to particular constituencies is taken into consideration in the proprietary software model when deciding about value-creating investments. Efficiency is hence impaired in this business model, and the system of open source seems to be the superior organizational design.

Are mixed configurations possible? Several companies adhering to the traditional, closed-source model of software production have reacted to the threat of competition from the open-source business model by adjusting their organizational design. Traditional software companies have begun to selectively disclose some part of their source code while retaining the copyright on it. Microsoft, for example, has started a Shared Source Initiative. With this initiative it reveals parts of the source code to governments, universities or some carefully chosen companies. But this strategy is not to be mistaken for open source. The software remains proprietary even if the source code is made publicly available. Control over the use of a software program, its distribution, and its modification is retained under the Shared Source Initiative. It is not possible to fork the software and release a modified version without restrictions.

Can such a strategy succeed and increase efficiency relative to a pure open-source model? Several theoretical arguments and the anecdotal evidence in the case of the Shared Source Initiative speak against that. Applying the ideas of systems theory, it is clear why this is the case. Being open in production but retaining the final product proprietarily creates an imbalance in both the power distribution and the incentive structure. Closed-source systems are never able to emulate the incentive structure of open source. The incentives to undertake specific investments are out of balance if the balance of power to appropriate economic rents from the final product is skewed in favor of that constituency which holds the copyright on the software program. Other constituencies are in such a setting reluctant to invest their time and effort into helping with the development of the software. Changes in other organizational parameters would be required as well, one of the essential changes being the configuration of the transparency of individual contributions to the final product to be high. This feature is, however, deliberately muted in

\[31\text{It should be noted that an economic rent can also be due to network effects and the prevalence of a common standard. This usually favors proprietary software where such effects are explicitly considered in the strategy of closed-source companies.}\]

\[32\text{See also West (2003) for further examples of hybrid strategies or Dinkelacker and Garg (2001) for a description of Hewlett Packard’s response to open source.}\]
firms. Having the best of both worlds is not going to work: It takes more than revealing the source code to obtain the benefits of open source. A strategic choice about the business model, such as the one made by Netscape Communications to release its Navigator web browser into the open source, has to be made.

5.2 Efficient organizational design

The arguments above might lead to the temptation to completely disregard the traditional model of software development and to advocate a disclosure of the source code as a panacea. This, however, would be a bit shortsighted. Answering the question which one of two systems is actually better is no trivial task. One caveat concerns exogenous factors such as the rate of change. Another applies when considering the efficiency of the larger organizational design which constitutes the overall economy.

Change and innovation revisited Bazaars are dismantled in order to be rebuilt every day, while cathedrals built long ago have weathered out many storms over time. Bazaars are often built around cathedrals. The existence of a cathedral summons merchants from different regions and allows them to coordinate their actions. Cathedrals provide focal points. The architectural metaphor fits the organizational design well: Like cathedrals, companies operating under the traditional business model with proprietary software have vested interests in resources and positions. They have erected rigid structures and often impermeable boundaries to protect these resources and positions and the economic rents generated by them. Dissolution is not considered. The power distribution in companies can become skewed over time in favor of particular constituencies, thereby discouraging value-creating specific investments by other constituencies that lack an offsetting power. The configuration of the various organizational parameters applied by commercial companies thus ossifies as time goes by and hinders them to deal with disruptive changes.

The flexibility inherent in open source allows it to cope better with disruptive changes. Radical adjustments are facilitated as projects can be initiated or discarded easily. A high degree of changes in the rugged landscape of competition therefore favors the organizational design of open source. Disruptive change happens, unless it is the idea of a single brilliant mind, mainly because of closely coordinated innovation. A system which ensures that no constituency can appropriate economic rents generated by specific investments of other constituencies encourages value creation. A thoroughly designed system of open source does this.

But recall that change and the desire to catch up with new products was one of the motivations behind the open-source system. If such change is absent and one crucial parameter has thus been altered to a configuration inconsistent with that of other elements, the whole system of open source can fall apart. As the software industry matures and the rate of change subsides, a model based on private innovation could become more appropriate again. Incentives to increase the share of economic rents through expropriation of rents from other constituencies are never muted. Even in open source, such behavior remains rational for individual parties and, if it is possible, will occur. Although the system of open source is probably the higher peak in the rugged landscape of competition, it is also the one with the steeper precipice. Small deviations from the optimal configuration of a few parameters are already enough to push the whole system over the edge into an abyss. As a high rate of change is an important parameter in the configuration of open source, the fragile design of open source is in danger when that rate is low. A general answer to the question whether one system is better than the other is therefore not feasible.
Twin peaks and efficiency of the larger design   It is, instead, more likely that the interplay of an open-source with a closed-source system achieves the maximum efficiency in the larger design of the overall economy than the sole prevalence of one particular system. The coexistence of both business models keeps the power of incumbent software firms adhering to the proprietary model and their strategic entrenchment in check. Companies operating under the traditional model of private innovation cannot exercise their power to the degree possible in a system in which open-source projects do not exist and other profit-oriented companies would not undertake the necessary investments to counter the power of a quasi-natural monopolist.

Open-source projects on the other hand would not exist if there was no competition from proprietary software corporations. Like bazaars need cathedrals to be built around, open source needs a functioning system with commercial software companies. It is in this case, however, not the issue of keeping a monopoly position under control. If some constituency in open source uses its power in a way to appropriate rents from other parties, it is easy to fork the project with an open source code. The problem with a design in which only open source communities prevail is that programmers in an idealized open-source system would lose the motivation to work for the commons. Free-riding on the effort of others would be more widespread. With little competition there would be little focus in the projects, and the signalling rationale for able programmers would break down. Thus deprived of an important configuration, the design would become inconsistent and would arguably fall apart. Open source hence needs closed-source companies to reinforce its merits as is also argued by Lee, Moisa, and Weiß (2004).

From the point of view of maximizing the welfare in the system of the overall economy, the choice is then between an inefficient system with a skewed balance of power and one that would not live up to its theoretical promises. But for the task of optimizing the design of the overall economy a third possibility exists. A coexistence and co-evolution of both systems of software production achieves efficiency in the overall economy. Competition between the two systems mitigates the potential weakness of each system on its own. Favoring the one or the other of these two idealized systems and switching between them as time goes by allows the avoidance of the weaknesses that would prevail when settling into the steady-state of applying on system only.33

The relative prevalence of the two distinct designs of software production determines the size of the commons area. If the commons domain is too large, no incentives to innovate exist due to the absence of a suitable appropriation regime. If it is too small, incompatibilities between competing standards lead to inefficiencies in design. Balancing collective and private innovation thus appears as the most efficient organizational design for the larger system of the overall economy. A rugged landscape of competition with two peaks, one peak representing the proprietary software system, the other peak the open-source system, is the suitable metaphor to describe the organizational design that maximizes the efficiency of the software industry in the overall economy.

References


33See Nickerson and Zenger (2002) for this argument in the context of organizational design in general.


