Osterloh, Margit; Rota, Sandra

Working Paper

Open Source software development - just another case of collective invention?

CREMA Working Paper, No. 2005-08

Provided in Cooperation with:
CREMA - Center for Research in Economics, Management and the Arts, Zürich

Suggested Citation: Osterloh, Margit; Rota, Sandra (2005) : Open Source software development - just another case of collective invention?, CREMA Working Paper, No. 2005-08, Center for Research in Economics, Management and the Arts (CREMA), Basel

This Version is available at:
https://hdl.handle.net/10419/214322

CREMA

Center for Research in Economics, Management and the Arts

Open Source Software Development – Just Another Case of Collective Invention?

Margit Osterloh

Sandra Rota

Working Paper No. 2005 – 08

Open Source software development – just another case of collective invention?

by

Margit Osterloh*
University of Zurich
Institute for Organization and Administrative Science
Plattenstrasse 14
CH-8032 Zurich
osterloh@iou.unizh.ch
Phone: +41 (0)1 634 28 40
Fax: +41 (0)1 634 49 42

Sandra Rota**
University of Zurich
Institute for Organization and Administrative Science
Plattenstrasse 14
CH-8032 Zurich
srota@iou.unizh.ch
+41 (0)1 634 29 39
Fax: +41 (0)1 634 49 42

* Margit Osterloh is Professor of Business Administration and Organization Theory at the Institute for Organization and Administrative Science, University of Zurich.

** Sandra Rota is a Research Assistant at the chair of Margit Osterloh at the Institute for Organization and Administrative Science, University of Zurich.

Open Source software development – just another case of collective invention?

Abstract

Does Open Source (OS) represent a new innovation model, and under what conditions can it be employed in other contexts? A look into history shows that OS isn't a unique example of what is called "collective invention". Other examples are blast furnaces in Britain's Cleveland district, steam engine design, and more recently, the flat panel display industry.

While OS shares many similarities with these cases of collective invention, there is a main difference: Other collective invention regimes did not survive after the development of a dominant design. It is argued that two factors can explain the difference. *Firstly*, OS licenses are important institutional innovations that make OS survive as a common property. *Secondly*, the second order social dilemma, which arises when it comes to developing and enforcing OS licenses, is overcome by the existence of intrinsically motivated contributors. It is asked under which conditions this kind of motivation is developed and sustained. We argue that it is the existence of certain self-governance-mechanisms which OS licenses are a part of. We conclude that OS differentiates itself from other cases of collective invention by its success in solving the second order social dilemma of rule development and enforcement.

(196 words)

Key words: Open source software, collective invention, intrinsic motivation, copyleft, conditional cooperation.

1 Introduction

Does Open Source (OS) represent a new innovation model, and under what conditions can it be employed in other than the software development context? OS is a term for software published under licenses that do not give any private intellectual property rights to the developers. They invest private resources in inventions that are then fed into a common pool by contributing to a public good.

A look into history shows that OS isn't a unique example of an innovation model which Allen (1983) called "collective invention". In the second half of the nineteenth century iron-making companies in Britain's Cleveland district willingly shared their innovations in blast furnace design. Other examples include the enhancement of steam engine design after 1800 (Nuvolari 2002) and the search for a dominant design in the flat panel display industry (Spencer 2003).

Von Hippel and von Krogh (2003) argue that OS represents a novel innovation model they call the "private-collective" model of innovation. It shares many similarities with the "collective invention" model. However, collective invention regimes did not survive after the development of a dominant design (e.g. Meyer 2002) while OS still is alive in such a phase of the innovation process.

This paper addresses three research questions: (1) Why did "collective invention" rarely survive after the development of a dominant design? (2) What are similarities and differences between OS and other cases of collective invention? (3) What are the conditions under which the "private-collective" model of OS survives after a dominant design has emerged? To answer these questions we give a short overview over the main characteristics of OS in the *second* section of this paper. In section *three* we present three other examples of collective invention. In section *four* we explore the common ground of the different cases. We discuss why collective invention is a phenomenon in the wake of a radical innovation that normally does not survive during the whole innovation process. In section *five* we compare these findings with the OS innovation process. We argue that OS licenses, which are grounded in copyrights, enable the collective invention regime to survive in a copyrighted world. To understand better how OS licenses help to reach that goal we describe the mix of selective benefits that coexist in OS similar to other collective invention cases in section *six*. We then ask what

conditions exist that prevent OS from changing from a “private-collective” innovation model to a “private investment” model. We argue that the characteristics of OS that are often identified as crucial, namely to be (a) an information product, (b) a user innovation that is (c) produced with a highly modular design, cannot explain fully the continuing survival of the collective invention model. But they enable what we call a low-cost-situation which facilitates contributions to public goods. We argue that only the interplay of low-cost-situations, OS licenses and governance mechanisms that foster and maintain intrinsic motivation can explain the success of the OS innovation model. We *conclude* with some remarks concerning the future of this model.

2 Characteristics of ‘open source’

OS is a very successful innovation model that challenges conventional economic views that innovation is best supported by strong private intellectual property rights (e.g. North 1981). In OS, programmers place their software at the free disposal of other users and developers. The software is published on the internet and anybody interested can download it for free. Users are not only allowed to use the functionalities of the software, they are also permitted to view the source code of the programs. [1] If they wish to change, debug or develop the code further, they are free to do so. They are even free to publish these amendments together with the original or the changed source code. In this way, some projects managed to attract communities of thousands of programmers who jointly develop the source without having exclusive property rights on the products of their work.

[1] Source code is the human readable form of software. To transform the code into a form that is readable for a computer, the source code has to be compiled into a binary version, i.e. a sequence of computer instructions consisting only of strings of ones and zeros. Usually when one buys proprietary software, what one gets is only the compiled, binary version of the program. The computer can read these instructions and deliver the functionality of the program, but for human beings it is basically impossible to read and understand. With open source software, one gets the binary version and the source code of a program.

In its essence, the term open source describes software licenses that explicitly give users the above mentioned rights.[2] Nowadays, there are a dazzling number of different OS licenses. The most important differences refer to the extent to which they allow public property to be mixed with private property rights. One of the most far reaching is the GNU General Public License (GPL). It forces every program that contains a free software component to be released in its entirety as free software. In contrast to the conventional copyright, this license is called "copyleft". It "infects" the OS software with a "virus" to enforce compliance to the copyleft. Thus, it is ensured that any derived software will remain free software. Other licenses, like Apache's, allow programmers to make their modifications private and distribute them as proprietary products. But even the least restrictive OS licenses ensure that what once enters the public domain cannot be re-appropriated as a private good.

The really puzzling feature about OS is that these programs partly constitute a public good in the classical sense (Lerner and Tirole, 2002). Once the source is published, nobody can be excluded from using it. Public goods usually face two problems: the overuse and undersupply of common resources. This is what Hardin (1968) called the "tragedy of the commons". In OS software production, the problem of overuse does not occur since there is no rivalry in consumption. Additional users can even generate positive external network effects. Nevertheless, this cannot explain why the problem of underprovision seems to have been overcome.

Von Hippel and von Krogh (2003) argue that OS represents a novel innovation model they call the "private-collective" model of innovation. The distinguishing mark of this innovation model is that economic actors invest their private resources to produce a public good. The authors argue that this model does not fit in one of the two major models that try to explain how incentives to invest in innovation can be furthered, i.e. the private investment model and the collective action model. The *private investment model* states that private investment in innovation is furthered if inventors can

[2] The open source definition includes some other defining features that are not of great importance for the arguments of this paper. To view the full definition, see e.g. O'Reilly and Associates (1999).

appropriate the returns on their investment. To this end inventors will try to avoid knowledge spillovers as far as possible and society may grant intellectual property rights to the inventors in the form of patents, copyrights, and trade secrets. The *collective action model* applies to the production of public goods like e.g. basic research. For this model to work, some central agent (e.g. the state) has to grant selective incentives in the form of e.g. monetary or reputation subsidies. Careful selection of contributors is important here, so that only individuals who are sensitive to these selective incentives are chosen.

Von Hippel and von Krogh (2003) show that OS resides in the middle ground between these models of innovation. Programmers freely reveal their innovations without claiming any private intellectual property rights. There is no central agent giving out selective incentives nor is there the hope of monopoly rents due to innovation. Contribution has to be self-rewarding, i.e. only programmers will contribute whose utility is greater when contributing than when free-riding. In that way, OS seems to unite the benefits of the private investment and the collective action model while avoiding their downsides.

Under what circumstances is this “private-collective” model of innovation applicable? Most answers to this questions point to three characteristics of software production that greatly facilitate this kind of joint innovation: (a) since software is an information product, innovation and production are essentially the same thing, (b) users are innovators, and c) the modular architecture of software design enables a decentralized production.

(a) Information products: There are two main characteristics that make a difference between physical and information products. *Firstly*, with physical products, production and distribution typically involve significant economies of scale. Thus, these activities are usually centralized to some degree and carried out by manufacturing firms (von Hippel 2001). For information products like software, reproduction and distribution costs are close to zero. This greatly facilitates joint innovation by a geographically scattered group of developers without the involvement of manufacturers. *Secondly*, with information products there is no rivalry in consumption (Baldwin and Clark 2003). Giving

information away doesn't preclude consumption by the donator. This property of non-rivalry in use basically applies to all non-physical objects, including ideas, designs, etc. Thus, one could conclude that the "private-collective" innovation model of OS could apply for all innovation projects since, at least before the production phase, all innovation is a non-physical good. The only restriction here is that contribution still has to be self-rewarding, i.e. the benefits of contribution must exceed the benefits of keeping an innovation secret (von Hippel and von Krogh 2003). This conclusion however is way too general and needs further elaboration. While innovation in itself is definitely a non-rivalrous good, the rents one hopes to generate with innovation are not. If an innovation is diffused widely, the innovator loses his monopoly position and his rents might well deteriorate.[3] These opportunity costs are the reason why a manufacturer will usually not share his insights with his competitors. For user innovators, as we will show in the following, these opportunity costs are often much lower, making sharing more likely.

(b) User innovations: Users can be a rich source of (especially incremental) innovation (von Hippel 1988). By working with products they get to know them intimately, detect their shortcomings and develop ideas how the products could be altered to better suit their needs. They are, however, often ill-positioned to market their innovations. Even the licensing option is often quite unattractive due to high transaction costs of intellectual property management (Harhoff, Henkel and von Hippel 2003). This leaves user innovators with two options: They may keep the innovation secret to exploit it in-house, or they decide to reveal it freely. In the case of *physical products*, a manufacturer might chose to incorporate the innovation in his product and end up offering the user innovator a better and cheaper solution than he could have developed in-house. In the case of *pure information products* like software, other users might build on the innovation and in turn release improved versions of the product that might again be useful to the original user innovator. In both cases, the user innovator

[3] This is not always the case though. In section 4 we will come back to circumstances where sharing of innovations can even enhance the profits of a manufacturer innovator.

needs to weigh these potential benefits against the costs of revealing. These costs are the lower (1) the more the revealed innovation is geared towards the specific needs of the user innovator and (2) the less competition there is between the user innovator and other potential adopters of the innovation (Harhoff et al. 2003).

(c) Modularity: Modularity is a way to manage complexity (Langlois 2002). A modular system consists of a complex of components or sub-systems where interdependencies among modules are minimized (Narduzzo and Rossi 2003; Sanchez and Mahoney 1996). Modularity enables many developers to work simultaneously on a project while at the same time keeping integration and coordination costs low.

While these characteristics of software production are definitely most important in explaining the huge success of some OS projects, a look into history shows that the "private-collective" model of innovation we see in OS isn't a new phenomenon.

3 Other examples of collective invention

In the following we will shortly present three cases that are similar to the "private-collective" model and compare them to the OS case. Comparable modes of innovation, called "collective inventions" (Allen 1983), could be observed in the development of blast furnace design between iron-making companies in Britain's Cleveland district in the second half of the nineteenth century. More recent examples include the Homebrew Computer Club (Meyer 2002) and the search for a dominant design in the flat panel display industry (Spencer 2003). Oddly, these other cases don't really seem to fit the above mentioned criteria for the applicability of the "private-collective" innovation model. In the following sections we will shortly present these three cases and compare them to the OS case. We will argue that the obvious differences between these cases can be reconciled if one takes into account the different stages of development of the respective technologies. We will conclude that the above mentioned characteristics of OS software are not fundamental conditions for the emergence of this innovation model but rather enabling conditions for its continuing survival.

3.1 Blast furnaces in Britain's Cleveland district (second half of the 19th century)

Allen (1983) was the first to analyze what he called "collective invention". Collective invention describes situations in which economic actors willingly reveal their innovations to an interested public so that others can learn and develop these innovations further. Technologies are thus developed by repeated interaction and feedback. Information is shared with the whole innovation system and not with just a selected few.[4] When looking at the development of blast furnace technology in Britain's Cleveland district in the second half of the nineteenth century, he discovered that the bulk of innovation neither stemmed from non-profit institutions like *universities* and *government agencies*, nor from R&D activities undertaken by *commercial firms* or *individual inventors*. Rather technological development was attributable to experimental incremental changes in design by firms whose results were then made available to other firms in the industry and potential entrants. He concluded that there was a fourth innovative institution which he termed *collective invention*. It led to dramatic technological changes in the blast furnace technology. Average height of furnaces increased from fifty to eighty feet, and the temperature of the blast increased from 600°F to 1400°F. These changes led to a significant reduction in fuel consumption. Sharing took place through informal disclosure and publication in the engineering literature. In 1869 the Iron and Steel Institute was established which served as a forum for the presentation and exchange of technical material.

According to Allen (1983) these firms didn't engage in formal research efforts. At that time, the chemical and physical aspects of blast furnace technology were not understood well. Nobody could predict what changes in productivity could be expected from design changes. Progress had to take the form of experimental trial and error learning. By sharing experiences firms could multiply the number of trials they could learn from, thus enhancing overall technological progress. But why

[4] Formally agreed-upon R&D cooperations and other contractual agreements like e.g. cross-licensing do not fall under collective invention.

should an individual firm acting in a competitive environment share relevant information and thus reduce its competitive advantage?

Allen sees three enabling factors for the emergence of this behavior. *Firstly*, it was difficult to keep the information private anyway. Engineers and other employees who were involved in building the facilities moved to other plants and took their experiences with them. Since the developments merely consisted of slight changes of existing designs they were not patentable. *Secondly*, entrepreneurs were engaging in some kind of competition who was trying out the most daring designs. They might have been willing to release information even if this meant sacrificing some profit. *Thirdly*, most entrepreneurs also were owners of ore deposits in Cleveland. Enhancing the efficiency of ore refinement technology enhanced the value of this complementary asset. Thus they were not only interested in the profitability of their own plant but also in the overall efficiency of iron ore refinement in the region. All of these factors enabled the emergence of a mutually beneficial sharing regime (Meyer 2003).

For Allen the most important lesson from his study is that collective invention can, to some degree, substitute for private R&D investment. But he suspects that nowadays, with the increasing importance of private R&D, collective invention should be less important. However, as the following two cases and the example of OS show, this phenomenon is still widely observable.

3.2 The Homebrew Computer Club (1975-1986)[5]

The Homebrew Computer Club was formed at Stanford University in 1975, after the invention of the microprocessor but before there was a personal computer industry. It was a meeting place for people who had become interested in building applications for microprocessors, thus exploring their potential. The group was open to everyone interested and it published a free newsletter. At club meetings, members would present their latest developments and discuss views about design philosophy. It was a meeting ground for people who shared interests and numerous companies were formed. Steve Wozniak was one of them. When he built his own computer, he brought it to one of the

[5] The following chapter draws on Meyer (2003).

meetings and passed around blueprints of its design so that others could copy it. He called his computer an Apple. Later he, together with Jobs and some other Homebrew members, would form the Apple Corporation.

With the emergence of marketable products, the atmosphere at the club changed. People who had formerly willingly shared information and who had only been interested in learning from each other were suddenly in the position of competitors who had to hide company secrets from each other. They stopped going to the meetings and the club slowly lost its appeal. It disbanded in 1986.

3.3 The flat panel display industry (1969 – 1989)

The first technological breakthroughs in flat panel display technology took place in the late 60s. But it took 20 years until one product design, the liquid crystal display, emerged as the dominant design for laptop computer applications and large-scale manufacturing and commercialization began. Spencer (2003) studied the knowledge sharing strategies of firms in the flat panel display industry in this pre-commercial phase between 1969 and 1989. She discovered that 80 of the 115 firms active in this industry published at least one scientific paper during this time. Spencer shows that firms that shared relevant knowledge with their global innovation system earned higher innovative performance than firms that did not share (measured by the value of their patent portfolio). Even though a causal relationship between the decision to share and innovative performance cannot be established, her data suggests that firms that employ the best scientists tend to have high numbers of publications and high innovative performance.

4 In search of the common ground of the three cases of collective invention

In all of these three cases of collective invention the actors involved were faced with a radical innovation which is a time of high technological uncertainty and experimental learning processes.

A radical innovation is a technological development whose potential isn’t well understood yet. While incremental innovations introduce relatively minor changes to existing products, exploiting the potential of the established design, radical innovations are based on a different set of engineering and scientific principles and often open up whole new markets and potential applications (Dess and Beard

1984). These periods of experimentation are brought to an end by the emergence of a dominant design (Abernathy and Utterback 1978). A dominant design is characterized both by a set of core design concepts (embodied in the components) and by a product architecture that defines the way in which these components are integrated (Clark 1985). After the emergence of the dominant design, the level of experimentation drops significantly and technological development takes place at the component level within the framework of a stable architecture (Abernathy and Utterback 1978; Henderson and Clark 1990).

Technologies typically evolve along so called technological trajectories (Dosi et al.1988). Radical technological breakthroughs cause disruptions in these trajectories. During an initial 'era of ferment' (Anderson and Tushman 1990), economic actors experiment with the new technology and rivalry centers on differences in the fundamental designs of the products (Teece 1987). This is the *exploration phase* of a technology in which the potential of a radical innovation is explored. Typically, only one of these different trajectories will emerge as the dominant design for any given end application (Spencer 2003). After that, the nature of competition changes and focuses more on incremental refinements within one trajectory (Abernathy and Clark 1985). The technology enters its *exploitation phase*. During the exploration phase, firms cannot rely on their own stock of competences but have to tap into external sources of knowledge (Stringer 2000). This is the case for *two* reasons. *Firstly*, cognitive distance is typically greater between firms than within firms. This variety of cognition is a prerequisite to create novelty and to explore the potential of the new technology (Nooteboom 2000). *Secondly*, firm's approaches often lie dispersed across distinct technological trajectories (Spencer 2003). Careful observation of other's approaches reduces the risk of getting stuck on a trajectory that ends up not being selected as the dominant design.

While it is rather clear why firms can profit from external sources of knowledge, it is less evident why they should be willing to share their own knowledge. In the alliance literature, knowledge sharing across firm boundaries is often represented as a social dilemma (see e.g. Larsson et al. 1998). While all partners to the alliance would collectively fare best if everybody shared their knowledge,

individually they can fare even better if they manage to appropriate their partners' knowledge without contributing their own. If all partners follow this opportunistic strategy, the alliance will not be able to achieve its learning goals. There are three factors that can explain why during the exploration phase firms can overcome that social dilemma: (1) great learning potential, (2) low opportunity costs in the pre-commercial phase and (3) selective benefits.

(1) The severity of the social dilemma of knowledge sharing is reduced if the knowledge shared creates great synergies, i.e. sharing creates new knowledge that goes beyond the sum of knowledge stocks exchanged (Loebbecke, Fenema and Powell 1998). This is the case when sharing generates *great learning potentials*. Members of the Homebrew Computer Club stressed that there was so much to learn about the new technology that sharing their developments was the only way to explore its potential (Meyer 2003). Iron makers in Cleveland had to build huge blast furnaces in order to try out new design concepts. For them, sharing their experiences was the only way to validate their results and to figure out which trajectories were promising for further experimentation.

(2) The severity of the social dilemma is also reduced if the potential negative impacts are low. In the pre-competitive phase, *losses from sharing tend to be low*, since there aren't yet any marketable products fighting for market share. On the contrary, Spencer (2003) argues that for the success of a new technology it can be imperative that several players converge on the same trajectory, e.g. by agreeing on the same standards so that complementary products can endorse the new technology. This is facilitated if firms openly share their developments. Doing this, they might end up getting a smaller part of the pie than if they had kept their developments secret, once the technology enters the commercial phase. But there is a fair chance that the pie itself becomes bigger.

(3) Still the question remains why firms don't choose to sit on the side and simply watch what others are doing rather then actively share their own knowledge. This puzzle can be solved if one considers *selective benefits* that only accrue to parties actively engaging in knowledge sharing. If such selective benefits exist, a social dilemma can be transformed into a coordination game where several equilibria exist (Sen 1974). In all of the above examples of collective invention, such selective benefits could be

identified. They are as diverse as the cases themselves and include reputation benefits, gains through value enhancement of complementary assets, or pure enthusiasm about the potential of a new technology (see e.g. Allen 1983, Meyer 2003, Nuvolari 2003). One selective benefit that applies to all three cases is the *influence on one´s innovation environment*. To develop a market for a new technology it can be important that several players follow the same technological trajectory. Every firm has an interest that its own trajectory emerge as the dominant design, so that it can exploit its experience advantage. By sharing their knowledge, firms might be in the position to attract other players to their own trajectory (Spencer 2003). Sharing knowledge reduces barriers to entry onto the technological trajectory of a given firm, since it gives other players clues about what works, what problems they should expect and what methods might help to overcome these problems. Additionally, if a firm is successful in this endeavor, it can optimally profit from the developments of other firms that build on its own efforts, since it induced others to invest their efforts in areas where it has a high absorptive capacity.

These three factors together (great learning potential, low losses from sharing, selective benefits from shaping one's innovation environment) explain why collective invention is a phenomenon often observed in the wake of a radical technological development.[6] However, once a technology enters the commercial phase of competition, one would expect collective invention regimes to break down (Meyer 2003). After the emergence of a dominant technological trajectory, competition shifts towards differentiation within a given dominant design as companies fight for market share (Teece 1987). Appropriation concerns start dominating; *exploration* of the potential of a technology is replaced by *exploitation* of its commercial value.

5 Is open source simply another case of collective invention?

OS software development is often described as the most global and successful case of collective invention (Meyer 2003, Nuvolari 2002) or the "private-collective" model (von Hippel and von Krogh 2003). However, OS does not fit into the conditions analyzed that make a collective invention regime

[6] For further examples see e.g. Meyer (2003) and Nuvolari (2002).

likely: Once a technology enters the exploitation phase, sharing knowledge promises less potential for acquiring new knowledge and for influencing the environment. Rather, the negative impacts of knowledge sharing increase, in particular rivalry about the appropriation of rents earned with the product. Sharing is no longer self rewarding. OS is clearly in the exploitation phase. As Narduzzo and Rossi (2003) show, most OS projects refrain from trying out new architectural designs. They mostly adopt already existing architectures. Also the characteristics of user innovation and modularity clearly show that OS has outgrown its exploratory phase. User innovation is a characteristic of the exploitation phase of a technology when innovations are incremental, not radical. Modularity can only be achieved if a technology is very well understood (Langlois 2002; Ethira, Sendil and Levinthal 2004). The question arises why, in the case of OS, a collective invention regime could survive the transition from exploration to exploitation. To answer this question let us give a short account of the history of software development.

In the early days of computer programming in the 1960's and 1970's commercial software was a rarity and was usually sold bundled with computer hardware. Users developed their own software or hired programmers to write a particular program for a particular purpose. Most software was developed in academic and corporate laboratories by scientists and engineers. Since these programs were not intended for sale but rather were working tools, developers freely exchanged code they had written (von Hippel and von Krogh 2003, Bessen 2002, Narduzzo and Rossi 2003). This was facilitated by the establishment of the ARPANET in 1969 and later on by the internet.

It was also a time of heated discussions about how good software should be written. While Brooks (1975) was propagating a monolithic approach in which 'a few good minds' do the design of the whole project and only implementation of the code is more widely delegated, Parnas (1972) spoke up for a modular approach in which a project is divided in subparts that are coordinated via specified module interfaces. The success of Unix as the first modern operating system that was portable to different hardware platforms decided this discussion in favor of a highly modular approach.

Unix is a highly modular, scalable and portable platform. Its architecture is massively decomposed into independent modules that 'do one thing and do it well'. The history of its development goes back to the late 1960's and was mainly coordinated by AT&T Bell Labs. AT&T freely distributed Unix to universities and commercial firms. The releases included the source code of the software. This enabled users to contribute to its development. Developers at MIT and Berkeley were especially active contributors, but commercial firms also released their own diverging versions of Unix, optimized for their own computer architectures.[7] In 1983 AT&T decided to end the confusion between all the differing versions and combined various versions developed at other universities and companies into Unix System V Release 1, the first supported release.[8] This new commercial Unix release however no longer included the source code and was sold for 50'000$ per license.

Up to that point, software development was really just another case of collective invention. Software development shows astounding similarities to the above cases of collective invention. In an initial phase, up to the beginning of the 1980's, an actual software industry had not yet emerged. Code was shared freely among developers in university and commercial research institutes. Only with the advent of Unix with its modular, platform-independent architecture a standard and dominant design began to emerge. When the technology had reached certain maturity, commercial developers like AT&T became reluctant to share their code and various proprietary versions were released. Software companies were formed to distribute their own operating systems based on the Unix architecture, among them Santa Cruz Operation (SCO) and Sun Microsystems.[9] Appropriation concerns came to the fore and the collective invention regime received a severe jolt.

As a consequence, many programmers saw themselves excluded from the development of a product they had helped to create. As a reaction, in 1984, Richard Stallman started a new project called GNU, an acronym for 'GNU is Not Unix' (Narduzzo and Rossi 2003). The aim was to develop

[7] See www.unix.org/what_is_unix/history_timeline.html.

[8] See en.wikipedia.org/wiki/UNIX.

[9] See en.wikipedia.org/wiki/UNIX.

a free alternative to the existing commercial and proprietary Unix versions. To prevent that GNU suffered the same fate as Unix, Stallman developed the GNU Public License (also known as "copyleft"). This license states that people are free to read, use and change the software, as long as the modified code inherits the same freedom rights of the original code. In this way Stallman used existing copyright laws in a very unusual way: Rather than claiming property rights, he used copyright to prevent anybody from appropriating the code. The free software movement was born. The term "open source" was introduced in 1998 at a meeting of free software activists, since they felt that the older term "free software" inspired thoughts of "gratis" rather than of freedom of opinion (O'Reilly and Associates 1999). Thus, OS can be seen as a movement to keep collective invention alive even when the technology reached its commercial phase. It seems that copyleft is an ingenious institutional innovation that makes the difference to the other cases of collective invention. To understand how this could be achieved we now have to take a closer look at the motivational aspects of contribution to OS projects.

6 Motivations to contribute to open source software

There is a whole range of different motives that act as selective incentives for contribution. If the contribution to OS is self-rewarding, the social dilemma situation of knowledge-sharing is transformed into a coordination game where defection is no longer the dominating solution. This is the case when the benefits exceed the costs. In section 6.1, the co-existence of different selective benefits in OS is described. In section 6.2, we ask what might be the difference between motivations in OS and other cases of collective inventions.

6.1 Selective benefits to contribute to open source software

The question of what selective benefits accrue to contributors of OS projects that are not accessible to free-riders has been widely addressed by scholars interested in the phenomenon (e.g. Ghosh et al. 2002, Hars and Ou 2002). These benefits can be divided in two ideal types of motivation, intrinsic and extrinsic motivation (Frey 1997, Osterloh and Frey 2000). *Extrinsic* motivation works through indirect satisfaction of needs, most importantly through monetary compensation. *Intrinsic* motivation

works through immediate need satisfaction. An activity is valued for its own sake and appears to be self-sustained. The ideal incentive system for intrinsic motivation consists in the work contents itself. Intrinsic motivation can be enjoyment-based or pro-social (Lindenberg 2001). Enjoyment-based motivation refers to a satisfying flow of activity (e.g. Csikszentmihalyi 1975) such as playing a game or reading a novel for individual pleasure. With pro-social motivation the good of the community enters into the preferences of the individual such as charitable giving (Frey and Meier, forthcoming), organizational citizenship behavior (e.g. Organ and Ryan 1995) or voluntary rule following even if this is not in the own self-interest. (Tyler and Blader 2000).

As empirical work has found out, programmers contribute to OS software for intrinsic and extrinsic reasons. Extrinsic motives are benefits through extended functionality, reputation and commercial motives. Intrinsic motives are fun and pro-social motives.

Benefits through extended functionality. Contributors to OS can gain extrinsic benefits as user innovators or lead users who tailor the software to their own needs (von Hippel 1988, von Hippel and von Krogh 2003). They publish their amendments because other users might work with the amendments of the code, maintain and develop them. Because the costs of publication on the internet are low, it can pay off even if the expectations for helpful comments from other users are relatively low.

Reputation motives. Contributors can make money indirectly by signaling their ability in the OS community which can then be turned into money through employment by a commercial software company or through easier access to venture capital. It is argued that in OS projects reputation can be more easily made visible than in proprietary projects due to the system of files that list people who made contributions, and due to the public nature of mailing list archives (Lerner and Tirole 2002, Moon and Sproull 2000).

Commercial motives. Commercial service providers make money by selling support, services or hardware for OS (for an overview see Markus et al. 2000). The most prominent example is Red Hat that sells support, services as well as autonomous components. Other commercial firms like Hewlett

Packard sell hardware, like printers, and contribute add-ons, like printer drivers, to make their products work with OS software. Finally, companies like IBM contribute to OS software by making their hardware compatible with it. These firms are absolutely vital for the widespread adoption of this kind of software, because the inexperienced consumer gets reliable services and add-ons (Kogut and Metiu 2000).

Fun. Much evidence exists that for a lot of programmers the work itself is intrinsically rewarding, based on individual enjoyment. Many actors of the OS community emphasize that the most important motives for participation are fun, learning and the public display of one's abilities. Writing or debugging software is perceived as a „flow experience" (Csikszentmihalyi 1975). More than 70% of OS developers report that they lose track of time while programming (Lakhani et al. 2002). As Raymond (2001) puts it: „We're proving not only that we can do better software, but that joy is an asset" (see also Brooks 1995 and Torvalds 1998).

Pro-social motives. The OS community is often described as a gift-culture instead of an exchange culture (e.g. Raymond 2001). OS contributors report that they like the sense of "helping others" or "giving something back" to like-minded others (Faraj and Wasko 2001). People seem to reply to the entire group when answering an individual question (Wellman and Gulia 1999). People who contribute for normative reasons produce a public good of two different orders. *Firstly*, they contribute to the functionality and quality of the programs (first order public good). *Secondly*, they are engaged in keeping the source code open (second order public good[10]).

The co-existence of intrinsic and extrinsic motives is absolutely vital to the success of OS. Intrinsically motivated contributors help projects to gain momentum so that they become attractive for extrinsically motivated contributors (Franck and Jungwirth 2003). On the other hand, extrinsically motivated contributors help to drive these programs into the mainstream.

[10] For first and second order public goods see the next section.

6.2 What is special about open source software?

The mix of extrinsic and intrinsic motivation discussed might not be specific for OS. Though we don't have empirical evidence, it might also be existent in the other cases of collective invention. Besides, in other contexts there exists much empirical evidence that a lot of people contribute voluntarily to public goods or commons (e.g. Frey and Meier forthcoming, Ledyard 1995). The question arises what is special about OS so that the voluntary contribution to a public good continues after the emergence of the dominant design when in the other cases sharing of knowledge was no longer self-rewarding. The answer is that in OS the first and second order social dilemmas are better solved than in the other projects.

How the first order social dilemma is mitigated in open source projects by low-cost-situations

The first order social dilemma is mitigated without a central authority when the selective benefits that accrue to contributors of a public good are self-rewarding. This is the case as long as the (extrinsic *and* intrinsic) benefits exceed the costs of contribution. Even among intrinsically motivated donators, martyrs and saints are in short supply. Donators are more willing to contribute if the private opportunity costs are not too high (Rose-Ackerman 2001, p. 553). According to North (1990, p. 43) there is a downward sloping demand curve for moral concerns. The more costly it gets, the less people contribute. On the other hand, if there exists a low-cost-situation, many people contribute small bits to the public good so that the total amount of contributions rises considerably (Kirchgässner 1992, Kliemt 1986). In contrast to the other cases of collective invention, OS remains a low cost-situation for a considerable number of programmers even when the commercial phase is reached, due to the three characteristics of OS mentioned in section 2.

The characteristics of *information products* keep costs of diffusion low. Participants simply post their contributions on the appropriate internet site. Due to *user innovation* the losses stemming from sharing intellectual property rights by using an OS license are often low compared to the gains from the expected feedback by other participants. Thus, these characteristics help to keep the costs of revealing knowledge low. Because of *modularity* the costs of the production of the source code are

also kept low. A modular architecture invalidates 'Brooks' Law' that adding people to a late software projects makes it later (Brooks 1975). With a non-modular architecture, more people involved in a project mean higher coordination costs that can, in the extreme case, render marginal returns of manpower on productivity negative. Modularization makes possible useful contributions at reasonable costs of integration.

How the second order social dilemma is solved in open source projects: Open source licenses and voluntary rule enforcing

The second order social dilemma consists in developing and maintaining the rules of voluntary cooperation. Creating and enforcing a code of ethics is itself a public good and thus constitutes a social dilemma of a higher order: "Punishment almost invariably is costly to the punisher, while the benefits from punishment are diffusely distributed over all members. It is, in fact, a public good" (Elster 1989: 41).

Low-cost-situations raise voluntary donations. But even in a low-cost-situation people do not contribute to public goods if there exist no rules that prevent them from being exploited by others. Empirical evidence shows that many individuals contribute voluntarily to public goods as long as some other individuals contribute also. They are „conditional cooperators" (Frey and Meier forthcoming, Levi 1988). This cooperation however is rather unstable since there is always the risk of a golden opportunity. In the collective invention context, a golden opportunity might take the form of a development of such great stand-alone value that the inventor is not willing to share it but rather tries to appropriate its commercial value. If the inventor can successfully exclude others from its use, collective invention efforts will not be sustainable.

This is impressively illustrated by the case of steam engine development.[11] Steam engine technology was already widely used and experimented with in Cornwall in the first half of the eighteenth century to solve draining problems in coal mines. These early versions of the steam engine had a very high fuel consumption determined by the necessity of alternately heating and cooling the

[11] The following paragraph draws on Nuvolari (2003).

cylinder at every stroke. In 1769 James Watt conceived a solution to this problem by introducing a separate condenser. Watt received a patent on this invention which he and his business partner Boulton refused to license, thus frustrating the attempts of other engineers to improve on the design. Only after the expiration of Watt's patent they were able to improve the design in a collective trial and error process very similar to the development of iron furnaces in Cleveland. Until the middle of the nineteenth century, the fuel efficiency of steam engines was quadrupled in this way. Interestingly, entrepreneurs were unwilling to use patented technology during that time.

How do OS projects prevent such golden opportunities from disrupting the collective invention process? There are two factors, firstly the institutional invention of the OS licenses, in particular "copyleft", and secondly the fact that a sufficient number of intrinsically motivated contributors voluntarily enforce OS rules.

Open source licenses. When people contribute their developments to OS projects, they do not simply give up their intellectual property rights. Instead, they use their property rights to protect their developments from ever being appropriated. This is achieved by the use of OS licenses, in particular "copyleft". It forces every program that contains a free software component to be released in its entirety as free software. Thus, it is ensured that software that was derived as a public good will remain a public good. OS licenses hinder exploitation of voluntary donors and ensure "conditional cooperation". They impose on the virtual community of OS users what Hansmann (1980) calls the "non-distribution constraint" which is characteristic for non profit organizations. The "non-distribution constraint" is a major institutional precondition for voluntary donations to organizations. It is the reason why institutions like the Red Cross or most universities are governed as non-profit organizations. Also for commercial providers who are dependent on the goodwill of the developers, it is crucial to commit credibly to the "non-distribution constraint". Thus OS licenses enable the coexistence and fruitful cooperation between commercial providers and voluntary donators (Franck and Jungwirth 2003).

Voluntary rule enforcing. Rules which maintain conditional cooperation mean little if they are not enforced. Empirical evidence shows that the rate of contributions to public goods drops dramatically in repeated interactions if defectors are not sanctioned (Fehr and Fischbacher 2003). If communities want to protect their work from being exploited by hijackers, they have to actively engage in rule enforcement. Violation of norms may consist in improper application of the license, removal of copyright terms or highjacking OS software by including it in proprietary software. But it is important to note that not only legal but also informal rules are observed and enforced. OS licenses are not merely legal documents. Rather they are the codified expression of a norm of reciprocity. O'Mahony (2003), who studied the means OS communities use to protect their work, comes to the conclusion that OS licenses are not effective because of their legal qualities (in fact, as yet their legal validity has never been tested in court[12]), but because the community is willing to accept and put through the norms they conceive as good behavior. The common idea behind all of these licenses is that "there is no limit on what one can take from the commons, but one is expected at some time, to contribute back to the commons to the best of one's abilities" (O'Mahony 2003, p. 13). Thus, even if a license basically allows one to sell a proprietary version of an OS project without ever having contributed anything to the project, this doesn't mean that this is accepted behavior. KDE (K Desktop Environment) gives an impressive example how rules of cooperation can be enforced. KDE is a Windows-like desktop for Linux and other OS operating systems developed by the OS community. It is based on a graphical interface toolkit called Qt. Qt was developed by Trolltech, a commercial software firm. It did not comply with the requirements of OS (Stallman 1999). Even though the Linux community agreed that KDE was a technically excellent product, many members refused to endorse it because they didn't agree with the terms of the Qt license. These members started a

[12] But there is an ongoing legal campaign of the SCO Group (that has the rights to the Unix version AT&T sold to Novell in 1993) against all the users of Linux, claiming that Linux is contaminated with code actually owned by the SCO Group, see en.wikipedia.org/wiki/UNIX.

parallel project called GNOME that is distributed under „copyleft“. Finally Trolltech reluctantly relicensed their product. By now Qt is available under a “copyleft“ license.

Rule enforcement can be managed in virtual communities without a central authority only if there exist a sufficient number of pro-socially motivated people who are prepared to identify and sanction rule breakers. Why do people develop a pro-social commitment to certain projects? Three conditions are relevant (Deci and Ryan 2000): (1) autonomy, (2) feeling of competence and (3) social relatedness.

(1) *Autonomy*: External interventions crowd out intrinsic motivation if the individuals affected perceive them to be controlling. In that case autonomy and self-esteem suffer. In contrast, intrinsic motivation is crowded in if a person’s feelings of autonomy are raised (for empirical evidence see Frey and Jegen 2001, Frey and Osterloh 2002). In OS projects autonomy is high compared to proprietary projects. Contributors choose for themselves where and what they wish to contribute. One of the norms in OS communities is that work can not be enforced neither by a central authority[13] nor by monetary incentives (Himanen 2001).

(2) *Feelings of competence* grow when individuals understand what they are doing and when they feel responsible for the outcome. These feelings strengthen perceived efficacy. There is significant empirical evidence showing a positive relationship between perceived efficacy and levels of contributions to collective goods (e.g. Kollock 1998). Feelings of competence are enhanced by feedback that is not perceived as controlling. This is the case in OS through user innovation which implements rapid feedback cycles and enables concurrence of design and testing.

(3) *Social relatedness* is of special importance for prosocial motivation. It raises group identity that has proven to have a strong impact on the amount of contributions to common goods (e.g. Kollock

[13] It should be noted that governance rules in open source projects differ to a great extent (for an overview see Markus et al. 2000)

1998)[14]. The OS community is often described as a social movement that is guided to a high degree by a strong identification with common norms of freedom of information or of fighting against the dominance of large companies like Microsoft (e.g. Hertel et al 2003).

In OS projects, intrinsic motivation is not only strengthened on the side of the sanctioner but also maintained on the side of the sanctioned. It is remarkable that most kinds of sanctions are informal in nature but nevertheless have a significant effect on behavior. The primary vehicles are on-line mailing lists and bulletin boards. They constitute what is called a „court of public opinion" (O´Mahony 2003). Such an institution is well suited to maintain intrinsic motivation. *Firstly* it enhances visibility of norms of socially appropriate behavior. There exists much empirical evidence that instructions about what is socially expected raise contributions to public goods significantly (Sally 1995). *Secondly* informal sanctions have a strong expressive function that is very suitable not to alienate people from the community even if they are reprimanded (Ostrom 1990).

7 Conclusion

Open source software production is a highly successful innovation model. But it is, by far, not a singular case but rather one example of what Allen (1983) calls collective invention. The purpose of this paper is to inquire under which conditions collective invention can emerge and be sustained.

By comparing the historical development of OS to other established cases of collective invention we showed that knowledge-sharing with the wider innovation system can often be observed in the wake of a radical innovation. Before a technology enters its commercially viable phase individual losses from sharing are rather low compared to the potential collective learning benefits.

Once a dominant design emerges, however, collective invention regimes usually receive a severe jolt. Development efforts shift from exploration to exploitation of a technology within the limits of an established architectural design. Profit motives crowd out knowledge-sharing based on reciprocity.

[14] Even a minimal definition of groups (e.g. those who prefer Kandinsky over Klee) has been found sufficient to create a group identification (e.g. Tajfel 1981).

In the case of OS this process has not taken place. OS is in its commercial phase but still exists – at least partly – as a collective invention. We have argued that to explain the difference between OS and other cases of collective invention, motivational aspects have to be taken into account. The maintenance of the collective invention model is *firstly* enabled by the characteristics of OS that create what we have called a low-cost situation in contributing to public goods. *Secondly* the institutional innovation of OS licenses maintain "conditional cooperation" of intrinsically motivated contributors. These licenses stipulate that what once enters the public domain cannot be appropriated privately. However, these rules mean little if its terms are not enforced. Rule enforcement in itself is a public good of a higher order. We argued *thirdly* that to overcome this problem there must exist a sufficient number of pro-socially motivated contributors who accept and put through the rules of the community. The governance structures of OS provide the conditions to foster that kind of pro-social motivation.

Thus the continuing success of the collective invention regime in OS heavily depends on a balance between selective intrinsic and extrinsic incentives to contribute to the first order public good (source code) and the pro-social intrinsic motivation of a sufficient number of participants who are willing to enforce the rules of cooperation. This balance however is threatened because legal protection of intellectual property rights on software is being strengthened in the USA and Europe (Bessen and Maskin 2000, Bessen 2002). Strong patent protection changes low-cost into high-cost situations. OS developers make no money with their programs so that they cannot afford legal fights (Osterloh and Rota 2004). So it will be important for government regulators to consider that stronger patent protection can damage the conditions of success of this innovation model and its applicability to other fields.

References

Abernathy, W. J., J. Utterback. 1978. Patterns of industrial innovation. *Technology Review* 40-47.

Abernathy, W. J., K. B. Clark. 1985. Innovation: mapping the winds of creative destruction. *Research Policy* **14** 3-22.

Allen, R. C. 1983. Collective Invention. *Journal of Economic Behaviour and Organisation* **4(1)** 1-24.

Anderson, P., M. L. Tushman. 1990. Technological discontinuities and dominant designs: a cyclical model of technological change. *Administrative Science Quarterly* **35** 604-633.

Baldwin, C. Y., K. B. Clark. 2003. *Does Code Architecture Mitigate Free Riding in the Open Source Development Model?* http://opensource.mit.edu/papers/baldwinclark.pdf.

Benkler, Y. 2002. Coase's Penguin, or, Linux and the Nature of the Firm. *The Yale Law Journal* **112(3)**

Bessen, J. 2002. What good is free software? R. W. Hahn (ed.) *Government policy toward open source software*. Brookings Institution Press, Washington, DC, 12-33.

Bessen, J., E. Maskin. 2000. *Sequential innovation, patents and imitation*, MIT Economics Department Working Paper, Cambridge, MA.

Brooks, F. P. 1975. *The Mythical Man-Month. Essays on Software Engineering*. Addison Wesley, Reading, MA.

Clark, K. B. 1985. The interaction of design hierarchies and market concepts in technological evolution. *Research Policy* **14 (5)** 235-251.

Csikszentmihalyi, M. 1975. *Beyond boredom and anxiety*. Jossey-Bass, San Francisco, CA.

Deci, E. L., R. Koestner, R. M. Ryan. 1999. Meta-Analytic Review of Experiments: Examining the Effects of Extrinsic Rewards on Intrinsic Motivation. *Psychological Bulletin* **125(3)** 627-668.

Deci, E. L., R. M. Ryan. 2000. The "What" and "Why" of Goal Pursuits: Human Needs and the Self-Determination of Behavior. *Psychological Inquiry* **11(4)** 227-268.

Dess, G. G., D. W. Beard. 1984. Dimension of Organizational Task Environments. *Administrative Science Quarterly* **29(1)** 52-73.

Dosi, G., C. Freeman, R. Nelson, G. Silverberg, L. Soete. 1988. *Technical change and economic theory*. Pinter Publishers, London.

Elster, J. 1989. *The cement of society: A study of social order*. New York, NY.

Ethira, J., K. Sendil, D. Levinthal. 2004. Modularity and Innovation in Complex Systems. *Management Science* **50(2)**, 159-173.

Faraj, S., M. M. Wasko. 2001. *The web of knowledge: An investigation of knowledge exchange in networks of practice*, Florida State University Working Paper, Tallahassee, FL.

Fehr, E., B. Fischbacher. 2003. The nature of human altruism. *Nature* **425, 23 Oct**. 785-791.

Franck, E., C. Jungwirth. 2003. Reconciling investors and donators – The governance structure of open source. *Journal of Management and Governance* **7** 401-42.

Frey, B. S. 1997. *Not just for the money: An economic theory of personal motivation*. Edward Elgar, Cheltenham, UK.

Frey, B. S., R. Jegen. 2001. Motivation crowding theory: A survey of empirical evidence. *Journal of Economic Surveys* **15(5)** 589-611.

Frey, B. S., S. Meier. Forthcoming. Social comparisons and Pro-social Behavior. Testing "Conditional Cooperation" in a Field Experiment. *American Economic Review*.

Frey, B. S., M. Osterloh. 2002. *Successful management by motivation. Balancing intrinsic and extrinsic incentives*. Springer, Berlin, Germany.

Ghosh, R. A., R. Glott, B. Krieger, B. Robles. 2002. *Floss final report – Part 4: Survey of developers*. http://www.infonomics.nl/FLOSS/report/FLOSS_Final4.pdf.

Hansmann, H. B. 1980. The role of nonprofit enterprise. *Yale Law Journal* **89** 835-901.

Hardin, G. (1968): The tragedy of the commons. *Science* **162(3859)** 1243-1248.

Harhoff, D., J. Henkel, E. von Hippel. 2003. *Profiting from voluntary information spillovers: How users benefit from freely revealing their innovations.* MIT Sloan School of Management Working Paper.

Hars, A., S. Ou. 2002. Working for free ? Motivations for participating in open source projects. *International Journal of Electronic Commerce* **6(3)** 25-39.

Hertel, G., S. Niedner, S. Herrmann. 2003. Motivation of software developers in Open Source Projects: An internet based survey of contributors to the Linux kernel. *Research Policy* **32** 1159-1177.

Henderson, R. M., K. B. Clark. 1990. Architectural innovation: The reconfiguration of existing product technologies and the failure of established firms. *Administrative Science Quarterly* **35** 9-30.

Himanen, P. 2001. *The Hacker Ethic.* Random House, New York.

Kirchgässner, G. 1992. Toward a theory of low-cost-decision. *European Journal of Political Economy* **8(2)** 305-320.

Kliemt, H. 1986. The veil of insignificance. *European Journal of Political Economy* **2(3)** 333-344.

Kogut, B., A. Metiu. 2000. *The emergence of E-Innovation: Insights from open source software development*. Reginald H. Jones Center Working Paper, Philadelphia, PA.

Kollock, P. 1998. Social Dilemmas: The Anatomy of Cooperation. *Annual Review of Sociology* **24** 183–214.

Lakhani, K., B. Wolf, J. Bates, C. DiBona. 2002. *The Boston Consulting Group Hacker Survey*. http://www.osdn.com/bcg/BCGHACKERSURVEY.pdf, http://www.osdn.com/bcg/BCGHACKERSURVEY-0.73.pdf.

Langlois, R. N. 2002. Modularity in technology and Organization. *Journal of Economic Behavior and Organization* **49** 19-37.

Larsson, R., L. Bengtsson, K. Henriksson, J. Sparks 1998. The Interorganizational Learning Dilemma: Collective Knowledge Development in Strategic Alliances. *Organization Science* **9(3)** 285-304.

Ledyard, J. 1995. Public Goods: A Survey of Experimental Research. A. Roth, J. Kagel (eds.), *Handbook of Experimental Economics*. Princeton University Press, Princeton, NJ.

Lerner, J., J. Tirole. 2002. Some simple economics of open source. *Journal of Industrial Economics* **50(2)** 197-234.

Levi, M. 1988. *Of rule and revenue*. University of California Press, Berkeley, CA.

Lindenberg, S. 2001. Intrinsic Motivation in a New Light. *Kyklos* **54(2/3)** 317-343.

Loebbecke, C., P. Fenema, P. Powell. 1998. Knowledge Transfer under Co-opetition. T. Larsen, L. Levine, J. De Gross (eds.). *Information Systems: Current Issues and Future Changes*. Laxenburg, Austria. 215-229.

Markus, M. L., B. Manville, C. E. Agres. 2000. What makes a virtual organization work? *Sloan Management Review* **42(1)** 13-26.

Meyer, P. B. 2002. *Collective Invention in History and Theory: Bessemer Steel and Open-Source Software*. U.S. BLS working paper. http://econterms.com/pbmeyer/research.html.

Meyer, P. B. 2003. *Episodes of Collective Invention*. U.S. BLS working paper nr. 368. http://opensource.mit.edu/papers/meyer.pdf.

Moon, J. Y., L. Sproull. 2000. Essence of distributed work: The case of the Linux kernel. *Firstmonday* **5(11)**.

Narduzzo, A., A. Rossi. 2003. *Modularity in Action: GNU/Linux and Free/Open Source Software Development Model Unleashed*. http://opensource.mit.edu/papers/narduzzorossi.pdf.

Nooteboom, B. 2000. Learning by Interaction: Absorptive Capacity, Cognitive Distance and Governance. *Journal of Management and Governance* **4** 69-92.

North, D. C. 1981. *Structure and Change in Economic History*. Norton, New York, NY.

North, D. C. 1990. *Institutions, institutional change, and economic performance*. Cambridge University Press, New York, NY.

Nuvolari, A. 2002. *Collective Invention Ancient and Modern: A Reappraisal*. ECIS working paper. http://www.sussex.ac.uk/users/prff0/industrie%20studies/presentationAlessandro%202002.pdf.

Nuvolari, A. 2003. *Open Source Software Development: Some Historical Perspectives*. http://opensource.mit.edu/papers/nuvolari.pdf.

O'Mahony, S. 2003. Guarding the Commons: How Community Managed Software Projects Protect Their Work. *Research Policy* **32(7)** 1179-1198.

O'Reilly and Associates Inc. 1999. *Open Source: kurz und gut*. O'Reilly and Associates, Köln.

Organ, D. W., K. Ryan. 1995. A meta-analytic review of attitudinal and dispositional predictors of organizational citizenship behavior. *Personnel Psychology* **48(4)** 776-801.

Osterloh, M., B. S. Frey. 2000. Motivation, Knowledge Transfer, and Organizational Firms. *Organization Science* **11** 538-550.

Osterloh, M., S. Rota. 2004. Trust and Community in Open Source Software Production. M. Baurmann, U. Matzat, eds. *Trust and Community on the Internet*. Forthcoming

Ostrom, E. 1990. *Governing the commons: The evolution of institutions for collective action*. Cambridge University Press, New York, NY.

Ostrom, E. 2000. Crowding out citizenship. *Scandinavian Political Studies* **23(1)** 3-16.

Parnas, D. L. 1972. On the criteria for decomposing systems into modules. *Communication of the ACM* **15(12)** 1053-1058.

Raymond, E. S. 2001. *The cathedral and the bazaar*. O'Reilly, Sebastopol, CA

Rose-Ackerman, S. 2001. Trust, honesty and corruption: Reflection on the state-building process. *European Journal of Sociology* **42(3)** 27-71.

Sally, D. 1995. Conversation and Cooperation in Social Dilemmas: A Meta Analysis of Experiments from 1958 to 1992. *Rationality & Society* **7** 58-92.

Sanchez, R., J. T. Mahoney. 1996. Modularity, Flexibility, and Knowledge Management in Product and Organization Design. *Strategic Management Journal* **17** 63-76.

Sen, A. K. 1974. Choice, orderings and morality. S. Körner (ed.). *Practical reason: Papers and discussions. Blackwell*, Oxford, UK.

Spencer, J. W. 2003. Firms' Knowledge-sharing Strategies in the Global Innovation System: Empirical Evidence from the Flat Panel Display Industry. *Strategic Management Journal* **24(3)** 217-233.

Stallman, R. 1999. The GNU operating system and the Free Software Movement. C. Dibona, S. Ockman, M. Stone (eds.). *Open sources: voices from the open source revolution*. O'Reilly, Sebastopol, CA.

Stringer, R. 2000. How to manage radical innovation. *California Management Review* **42(4)** 70-88.

Tajfel, H. 1981. *Human Groups and Social Categories*. Cambridge, UK: Cambridge University Press.

Teece, D. 1987. Profiting from technological innovation: implications for integration, collaboration, licensing and public policy. D. Teece (ed.). *Competitive Challenge*. Ballinger, New York.

Torvalds, L. 1998. FM interview with Linus Torvalds: What motivates free software developers. *Firstmonday* **3(3)**.

Tyler, T.R., S. L. Blader. 2000. *Cooperation in Groups: Procedural Justice, Social Identity, and Behavioral Engagement.* Philadelphia: Hove Psychology Press.

Von Hippel, E. 1988. *The Sources of Innovation*. Oxford University Press, New York.

Von Hippel, E. 2001. Innovation by user communities: Learning from open source software. *Sloan Management Review* **42(4)** 82-86.

Von Hippel, E., G. von Krogh. 2003. Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science. *Organization Science* **14 (2)** 209 – 223.

Von Krogh, G., S. Spaeth, K. Lakhani. 2003. Community, joining, and specialization in open source software innovation: a case study. *Research Policy* **32(7)** 1217-1241.

Wellman, B., M. Gulia. 1999. Virtual communities as communities. M. A. Smith, P. Kollock (eds.). *Communities in cyberspace*. Routledge, New York, NY.