Explaining the organization of open source communities with the CPR framework

Ruben van Wendel de Joode
Delft, University of Technology (The Netherlands)
rubenw@tbm.tudelft.nl

Abstract

This paper describes work-in-progress. It describes the background, research framework and some preliminary results from a PhD research on the organization of open source communities. Most open source communities are very small. However, some communities have become very popular and they connect thousands of predominantly highly skilled programmers from various parts of the world. Together these programmers create and maintain highly complex software. Well-known examples of such communities are Apache and Linux. The software developed in open source communities has one very important characteristic: the source code of the software is open and freely available.[1]

To many it is highly surprising that programmers in open source communities are able to create successful software. Two questions prevail, they are: a) how are open source communities able to deal with internal pressures like free-riding and cascading conflicts and b) how are they able to resist external pressures, created by parties who appropriate software through copyrights and patents? This paper addresses the question how programmers in open source communities organize and sustain themselves amidst these pressures. Ostrom's (1990) eight design principles are adopted to answer this question.

The two most dominant conclusions from this research are: (a) individuals in open source communities are driven by individual choice and (b) formal mechanisms have a limited role in solving the issues addressed by the design principles. This article will analyze one design principle in more detail, namely the presence of conflict resolution mechanisms.

[1] The source code of software is the human-readable part of the software, which allows programmers to understand how the software works and allows them to modify the software if they choose to do so.

Introduction

Popular open source communities connect hundreds if not thousands of individuals who have different interests, backgrounds and motives. Together the individuals create highly complex software. Two of the best-known examples are probably Linux and Apache. Most of the individuals in open source communities are volunteers who are not paid to participate in open source communities, but the number of paid programmers is rapidly rising (Hertel, Niedner, & Herrmann, 2003). Much research has addressed the question why so many volunteers participate and donate their knowledge, time and effort in open source communities. Some of the frequently cited reasons are: to earn a reputation (e.g. Lakhani & Von Hippel, 2003; Raymond, 2001), want to learn (e.g. Von Hippel & Von Krogh, 2003), to solve a personal need (e.g. Wayner, 2000) or to have fun (e.g. Markus, Manville, & Agres, 2000; Torvalds & Diamond, 2001). Finally, many of the programmers in open source communities never get to see each other in real life; they meet virtually, in places on the Internet.[2]

The question that occupies many researchers is: given that volunteers are motivated how do they coordinate their activities? (Benkler, 2002a; Egyedi & Van Wendel de Joode, 2003; Egyedi & Van Wendel de Joode, 2004; Kogut & Metiu, 2001; McGowan, 2001; Moon & Sproull, 2000) How do they for instance deal with an issue like free riding, how do they prevent conflicts from endangering the continuity of a community or how do they decide what software to include and what to exclude? To many these questions are interesting because the communities are not engineered: there is no institution of some sort that has purposefully arranged the communities and their processes. Therefore some argue that the communities of programmers are examples of self-regulating and self-organizing communities. (e.g. Axelrod & Cohen, 1999)

Programmers in open source communities freely share their inventions, knowledge and ideas. One way in which they do so is by sharing the source code of software.[3] This source code is needed to understand and modify the software. This means that programmers from anywhere in the

[2] The metaphor of place on the Internet is frequently used (Hunter, 2003).

[3] The source code of software is "the human-readable instructions, which make up the software, before it is translated into something computer-readable (a binary file)… The source code for software may be compared to the blueprint of a construction. In order to use software source code has to be compiled. When source code is compiled, a special program (a compiler) translates the human-readable source code into a machine-readable code, which a computer can understand (Edwards, 2001, p. 4)."

world can download the source code of open source software and make changes to that source code. The fact that the source code is open and accessible to anyone has led many to argue that the source code is in the *commons* (Benkler, 2002a; Bollier, 2001; Bruns, 2000; Kogut & Metiu, 2001; McGowan, 2001).

Intellectual property rights like copyrights and software patents threaten the open source commons, as more and more companies use these rights to claim private ownership over software. (Benkler, 2002b; Boyle, 2003; Van Wendel de Joode, 2004b; Van Wendel de Joode, De Bruijn, & Van Eeten, 2003) In a way this process can be compared to a game of land grab:[4] companies and individuals are fencing off pieces of software. Organizations use *patents* to block others from using the ideas underlying their software and they use *copyrights* to block others from copying and using the actual lines of source code. It is this fencing or land grab, which is threatening the continuity of open source software. Intellectual property rights like copyrights and patents are effectively threatening the future flow of benefits from open source communities (O'Mahony, 2003). The challenge is to understand how open source communities are able to protect themselves against this threat.

Based on the above one is inclined to assume that software developed in open source communities will never be successful. Empirical data, however, suggest differently. The data show that some open source programs are quite successful in today's software markets.[5] First, some open source programs dominate their (niche) market. The program *Sendmail* is a good example. Sendmail is developed as an open source product and currently provides the standard method of routing e-mails on the Internet. In 1998, Sendmail was estimated to handle eighty percent of all e-mail traffic (Lerner & Tirole, 2002). Second, an increasing number of companies decide to support their

[4] This is adopted from Bessen (Bessen, 2002).

[5] This is, however, not the same as the claim that software developed in open source communities is qualitatively better than proprietary developed software. The quality of proprietary software or open source software will differ for each individual software program and the relationship between the quality of the software and the way in which the software was developed is all but understood. The only claim that is made here is that in certain niche markets open source software has been able to gain a decent share of the market and has apparently been able to reach a satisfactory level of quality, which in itself can be considered a surprise when we realize that the software is developed by a large number of volunteers who share their inventions. Essentially, this paragraph intends to demonstrate that open source software is not merely a hobby project of a large number of volunteers worldwide. Instead, it has become an increasingly important form to create and maintain software in today's market. The remainder of this section will demonstrate the success of a number of open source programs.

mission-critical business processes with open source software. One such example is the *New York Stock Exchange*.[6] Third, worldwide many governments and municipalities adopt open source software and recommend and sometimes even mandate the use of open source software. In May 2003 the city council of *Munich* in Germany decided to switch 14.000 desktops away from Microsoft to open source software. Currently, the city is in the middle of the transformation process.[7] Fourth, certain open source programs are of a comparable quality as proprietary developed software when measured by the number of defects per 1.000 lines of source code. A good example of a qualitatively comparable product is *Apache*, which is "found on par with commercial equivalents."[8] Research shows that Apache has 31 software defects in 58,944 lines of source code. This means a defect density of 0.53 per 1,000 lines of source code. This is comparable to the defect density of comparable proprietary developed software programs, which have an average defect density of 0.51.[9]

Problem question

The fact that software developed and maintained in open source communities has received so much attention, is being adopted by so many businesses and governments, is of a comparable quality, faces collective action problems and is threatened by external pressures created by the system of patents and copyrights, has resulted in the following problem question.

How do open source communities organize and sustain themselves in light of internal and external pressures?

---

[6] Based on an article on the Internet: http://www.it-director.com/article.php?articleid=2125 (November 2003)

[7] Based on an article in Het Parool (Dutch newspaper) by M. Laan: *Amsterdam is duur Microsoft zat*, October 14, 2003, p. 9. There are also many references to this project in articles on the Internet, for instance: http://www.wired.com/news/infostructure/0,1377,62236,00.html (March 2004)

[8] Cited from an article on the Internet: http://www.infoworld.com/article/03/07/01/HNreasoning_1.html (March 2004). The choice for the word commercial is somewhat unlucky, as open source software can also be commercial. Use of the word proprietary would be more appropriate. Proprietary highlights the fact that the owner develops software with exclusive rights and considers it his property, which is the most fundamental difference between open source and other types of software development.

[9] Based on the website: http://www.infoworld.com/article/03/07/01/HNreasoning_1.html (March 2004).

Research framework

Open source communities sometimes connect thousands of individuals, are geographically distributed and a human engineered product is developed and maintained in them. The theoretical framework adopted in this research, however, has its origins in an entirely different setting compared to the setting of open source communities and is based on research on common pool resources (CPR). This research typically focuses on relatively small communities that are primarily locally organized and the resources are primarily natural. (Ostrom, 1990, 1998, 1999) Examples of such communities are fishers who try to manage the scarcity in fishing grounds in Alanya, Turkey and farmers who have created a complex set of rules to manage their irrigation systems in Orihuela and Murcia, Spain (Ostrom, 1990). The results from research on common pool resources have increasingly been applied to bigger and more complex resources. One good example is research that focuses on what are called the "global commons." (Berkes, 2000; Buck, 1998) Furthermore, a number of researchers have applied the framework in a digital environment. Ostrom and Hess for instance apply the research framework to digital archives (Hess & Ostrom, 2001) and Bernbom has applied the framework to the Internet infrastructure (Bernbom, 2000).

The primary focus is on common pool resources (CPR). Economic thought basically states that without state intervention or without private property regimes, these types of resources will automatically face a depletion of the resource, which is also referred to as a "Tragedy of the Commons" (Bollier, 2001; Hardin, 1968; Ostrom, 1990; Tang, 1992). Research on community managed common pool resources has indicated, however, that this is not always the case; under certain conditions people have proven that they are able to organize themselves to produce and maintain a common pool resource. In these situations no outside intervention was needed to prevent a Tragedy of the Commons.

One of the first researchers to describe this was Elinor Ostrom. She wrote her findings, which are based on a great number of case studies, in a book called *Governing the Commons; The Evolution of Institutions for Collective Action* (1990). In the book, Ostrom builds a bridge between self-organization, collective action, common pool resources and a variety of property regimes. It turned out that many people had an interest in Ostrom's findings and soon a new school of thought started to form. A large part of this school consists of people that either work at or are associated with the Workshop in Political Theory and Policy Aanalysis at Indiana University (Ostrom, 1999). This group of people started to add much empirical evidence to the analysis of common pool resources and, although the book was written in 1990, its most fundamental ideas still apply.

Eight design principles

One of the contributions of Ostrom's book is the identification of eight so-called "design principles." According to her, these design principles can explain why certain communities are able to successfully coordinate the activities of individuals to manage a common pool resource and why other communities are not. In her book she is very cautious about the principles and claims that they are merely a first attempt of building a coherent set of design principles (p. 90). Yet, the principles have been adopted and lay the foundation in the work of a large number of researchers. (e.g. Anderies, Janssen, & Ostrom, 2003; Buck, 1998; Davies, 2001; Muchapondwa, 2002; Sarker & Itoh, 2001; Sekher, 2001; Tucker, 1998) The sheer number of researchers who adopted the design principles and who advocate their value is taken as an indication that the design principles are still relevant and have undergone no dramatic changes. This idea is further strengthened by the observation that Ostrom has continued to use and explain the same eight design principles. (Anderies et al., 2003; Ostrom, 2000) The eight principles are listed in table 2.1.

Table 2.1 - Design principles of long-enduring communities[10]

1.	Clearly defined boundaries
2.	Congruence between appropriation and provision rules and local conditions
3.	Collective choice arrangements
4.	Monitoring
5.	Graduated sanctions
6.	Conflict resolution mechanisms
7.	Minimal recognition of rights to organize
8.	Nested enterprise

[10] Adapted from Ostrom (1990). See pages 88 to 102 of the book for an elaborate introduction of the eight design principles.

Reasons to adopt the lessons from research on CPR

There are three reasons to adopt the lessons from research on common pool resources. First and most important, the question that is central in research on community managed common pool resources is how communities organize and sustain themselves. Contrary to popular belief, so the argument goes, the market mechanism and the state are not the two single ways in which coordination can be achieved, neither are they always the two most promising ones. (Ostrom, 1990) There are many situations in which communities of individuals are perfectly capable to organize themselves without the help of the state or the market. In that sense, open source communities can be compared to communities analyzed in the research on common pool resources, which are both excellent examples suggesting that the ability to organize is not restricted to the market or the state.

Second, research on common pool resources explains how communities are able to produce and maintain resources that are publicly available. This is comparable to open source software. However, CPR research is focused on common pool resources and software is a typical public good; it is hard to exclude potential beneficiaries and consumption is joint. Yet, it is argued here that the challenges facing open source software are comparable to the challenges facing common pool resources. The reason is that also in open source software the future flow of benefits are threatened (O'Mahony, 2003). Consider the influence of intellectual property rights, like copyrights and patents. They enable actors to claim private property on software and to fence them off from use by others (Bollier, 2001; Cowan & Harison, 2001; Van Wendel de Joode et al., 2003). The presence of these rights enables a form of appropriation of open source software, which is subtractable. Open source communities must find ways to protect themselves against this threat (O'Mahony, 2003). And therefore, although the resources are different, the challenges facing open source communities do resemble the challenges facing communities in which a common pool resource is managed.

Third, the eight design principles provide a coherent framework to analyze community managed common pool resources. In light of the current state of the art on open source communities, such a framework is relevant to be able to analyze and explain the organization of these communities. Until now, most research on open source communities (a) focuses on small parts of the puzzle or (b) merely provides some nice analogies.

Much research on open source communities focuses on one limited part of the organization of open source communities. In itself this is not the problem, however, they do not answer the problem question in this research. They do not explain the organization of open source

communities. One example is the leadership-induced model. Most projects have one or more clearly identifiable project leader(s) and they are suggested to be the primary and most important part of the organization of open source communities. (Bonaccorsi & Rossi, 2003; Fielding, 1999; Lerner & Tirole, 2002; Markus et al., 2000; McCormick, 2003; Moon & Sproull, 2000) However, leadership can at best be a part of the explanation (e.g. Van Wendel de Joode et al., 2003). Another example is the social space constructed by open source licenses. Open source licenses are the most important boundaries to shield open source communities against appropriation of the software. (O'Mahony, 2003) It appears as if some researchers consider this the single most crucial aspect and therefore the most important explanation of the organization of open source communities. (e.g. Bollier, 2001; McGowan, 2001) However, it must be clear that both explanations, the leadership-induced model and the social constructed by open source licenses, are just a part of the organization of open source communities and do not provide a complete answer to the problem.

There is also research, which is limited to the level of analogies. One frequently cited analogy is that of open source communities as self-organizing or complex adaptive systems (Axelrod & Cohen, 1999; Bekkers, 2000; Garzarelli, Forthcoming; Kuwabara, 2000; Madey, Freeh, & Tynan, 2002). In itself this is an interesting analogy and perhaps a very accurate one. However, it is not sufficient to explain an organization. Self-organization is not some form of magic. The challenges are to understand how self-organizing communities are able to deal with the collective action problem and are able to avoid a tragedy of the commons.

Another common analogy is that of open source communities as a gift economy (e.g. Bergquist & Ljungberg, 2001; Ljungberg, 2000; Markus et al., 2000; Raymond, 2000; Zeitlyn, 2003). The idea of the gift economy can be traced back to Mauss (1990), who described a wide range of communities in which gift giving laid the fundament of exchange. He argues that a typical gift economy relies on the principle of reciprocity and the implicit requirement to give (Mauss, 1990). Again, the critique arises, namely if open source communities are gift economies, how can we explain how they are able to deal with the collective action problem and prevent a tragedy of the commons. What are the mechanisms or principles that can explain the organization of open source communities? The analogy of the gift economy does not provide a sufficient answer to such a question (e.g. Iannacci, ; Weber, 2000).

Methodology

This research is qualitative and aims to understand and explain how open source communities are able to organize and sustain themselves. This section will describe the specific activities performed to provide an answer to this question.

Research strategy

To structure the analysis of the research question the eight design principles are adopted. They are believed to explain how communities are organized. The eight design principles are, if you will, the propositions that guide the research and enable the formulation of an answer. The primary goal of this research is to explain and the design principles in the form of proposition tell the researcher where to start looking for an answer. (Yin, 1989) This research will not be an exercise to verify the eight design principles and their applicability to open source communities or, which would be even more ambitious, to verify the applicability of the principles in communities on the Internet in general. Instead, they are used as guidance and a way to decide where to look and what to look for.

Applying the design principles does not mean that the absence of a design principle automatically leads to the conclusion that coordination is absent and that the future continuity of open source communities is under pressure. Primarily, it is taken to mean that the issue addressed in the design principle must be solved in some other way. Essentially, one could therefore argue that it is not the design principle itself, which is the proposition; it is much more the issue that is addressed by the principle, which is perceived to be crucial to explain the organization.

An example. Consider the principle: the presence of conflict resolution mechanisms. Essentially, the basic assumption that underlies this design principle is that people organized in self-organizing communities will run into conflicts, which must be solved if coordination is to be ensured. What does it mean if conflict resolution mechanisms are absent? One could be tempted to argue that the organization of open source communities is fragile, as they have no way of resolving conflicts. This, however, will not be the conclusion. The primary reason is that Hess and Ostrom (2001) already argued that the design principles in their current form are probably not applicable in a digital environment. Therefore, answers are sought to the question why these mechanisms are absent. For example, it could be that there is perhaps no need to solve conflicts in open source communities. The presence of other mechanisms might prove to explain why conflicts in open

source communities do not threaten the organization of open source communities. Another option could be that conflicts in open source communities are absent.

Data collection

Data collection has been performed in a number of ways. The most important source was the data from *interviews*. In total 60 interviews were held. Most of the respondents were programmers and contributors from at least one open source community. They constituted 37 respondents in total. Of them, 5 are or were project leaders of an open source community. Respondents, who are not programmers, had a different relationship with open source communities. The two most common groups are: managers or employers from companies that have adopted open source software but do not contribute to open source communities, 14 in total; and people who write about open source communities, 7 in total. Of the latter group 3 were doing their PhD research on open source communities and two were chief-editors for specific open source journals.

The interviews were semi-structured with open-ended questions. This means that they are a mix of elements from unstructured and structured interviewing techniques. The interviews were not structured nor were the questions closed. No questionnaires were used that have "preestablished questions with a limited set of response categories." (Fontana & Frey, 1994, p. 363) The interviews were neither unstructured, as the questions were not based on observations only. (Fontana & Frey, 1994) Instead, they were semi-structured: they focused on the issues addressed by the eight design principles. For example, related to the design principle "monitoring," the respondents were asked questions, like, what do you feel is counterproductive behavior? Does this type of behavior occur regularly? Why do you feel it is counterproductive? How do you know this type of behavior occurs? Do you regularly check if someone acted this way? Based on the answers from previous interviews, answers that were given during the actual interview, the background of the respondent, and the data collected from other sources, the respondents were asked more specific and detailed question about specific issues.[11]

Another source of data was *secondary literature*. Open source communities are heavily analyzed and researched objects of study. One of the primary sources for research papers, articles and books is the open source website hosted by MIT.[12] The site was created at the end of 2001 and currently

[11] This process is in line with the process described in Travers (2001)

[12] See: http://opensource.mit.edu/ (March, 2004)

lists the names of 444 people who are, for different reasons, interested in open source communities. Of these, 278 are connected to a university or another kind of research institute. The number of papers listed on the site is 147. Next to this massive amount of papers, there are an increasing number of articles in journals, books and conference proceedings. Many of these are used as an input for this research.

Direct observation was performed in a variety of locations and different settings. One important source of direct observation was at conferences, workshops and so-called user meetings. Two open source conferences were attended, namely the O'Reilly Open Source Convention in San Diego in July 2002 and the Open Source and Free Software Developers (FOSDEM) in Brussels in February 2001. Next to these two larger conferences, many smaller seminars in The Netherlands have been attended. One such meeting was a yearly members meeting of the Dutch Society for Open Source (VOSN) in Utrecht (The Netherlands) in 2001. The conferences and seminars were used to listen to the presentations that were given by a wide variety of people who are somehow connected to one or more open source communities. Furthermore, the interactions between the participants were observed. Field notes were made during and after the meetings.

A fourth source was the data collected from *archival records*. Many of the activities in open source communities take place on the Internet. These activities are usually archived. For instance, the actual upload of a new piece of open source software or the e-mails on mailing lists are both archived. These archives have been extensively analyzed. The most important source on the Internet that was analyzed, are the Internet resources that are directly linked to a particular community. Especially the weekly summary of the major happenings on the Linux kernel mailing list have been extensively studied in the period between November 2001 and June 2002. A working document was created in which observations were collected and grouped.

Data analysis

Data analysis started as a parallel trajectory of the data collection process. A large part of this process was to reduce and to make sense of the empirical data. A big part was the creation of a living document. This document was used to group the most recurring data from the data collection

phase. This grouping resulted in the identification of a great number of concepts.[13] These concepts were rather loosely defined and remained very close to the terms used by the respondents. (see for instance Spencer et al., 2003) One good example of such a concept is *elegance*. In a number of interviews, in discussions on mailing lists, and in secondary literature this concept was addressed. The challenge became to understand what the concept of elegance meant for the organization of open source communities. Does elegance have a role in the organization of the communities and if so, what is their role? In the following interviews the concept of elegance became a part of the interviews and the respondents were asked what elegance meant to them and what they perceived to be the role of elegance. This way a better image of elegance and its role in the organization was obtained. A similar process was used to better understand the role of other concepts like modularity, project leadership, voting systems, and open source licenses etc.

At the time the research proposal was finished (September 2001) eight tentative propositions were defined, which corresponded with the eight design principles. However, they were all formulated negatively. They were negatively, because a first exploration of open source communities appeared to demonstrate that none of the principles was actually present. One good example was the design principle clearly defined boundaries. Much literature argues that open source communities are open and membership is fluid. (Sharma, Sugumaran, & Rajagopalan, 2002) Therefore, one proposition was that boundaries were not present.

At the end of the data collection process every transcript and field note and the living document were analyzed. This analysis was structured along the eight design principles. To allow structuring of the findings, the design principles were first defined and made operational. Take for example the principle graduated sanctioning. A theoretical investigation was started to analyze what sanctioning is, how it is defined in literature and why it is needed. This enabled a definition of graduated sanctioning, which made it possible to identify which of the empirical data are related to sanctioning and which are not.

After having defined the principles, eight separate documents, for each principle one, were created and the data and the concepts were categorized along them. In some cases the data and concepts could easily be connected to just one of the design principles. In other cases, however, this

---

[13] These concepts are similar to what is known as *sensitizing concepts* (Blumer, 1954), which "give a general reference to empirical instances, later developing into more analytical, definitive concepts…" (Spencer, Ritchie, & O'Connor, 2003, p. 203)

process proved to be much less straightforward. There were two reasons to explain why. First, a number of concepts seemed to have a role in more than one principle. One example is the presence of *project leaders*. Should project leadership be categorized under the heading of collective choice or conflict resolution mechanism? In this example the remarks and notes about project leadership were categorized under both principles. Depending on the exact statements the remarks would be placed in one of the two documents.

Second, a number of issues were difficult to relate to a design principle. One example is the so-called *orphanage* in the Debian community.[14] Two respondents explained that Debian has an orphanage. The orphanage is a virtual place in which programmers can drop their package when they no longer, for whatever reason, have an interest in maintaining a certain piece of software. Other people in the Debian community can now take over the role of the programmer if they want maintain the software. Question is: what role does this orphanage have in the organization of Debian?

Two dominant images

From the research two observations clearly dominated all other observations. They are: (a) processes in open source communities are dominated by individualism and individual choice and (b) formally institutionalized mechanisms to aggregate decisions on a collective level explain little about the actual processes and practices in the communities.

The importance of individual choice

In the interviews the volunteers indicated that they behaved highly individualistically and based their choices on personal motives. Consider the statement from a maintainer of a number of software packages. He argues: "Well, you have a pool of people who do what they want to do. Nobody gives me an assignment. Instead, I think: "hey how weird, this process is very slow." Well, then I myself will start to work to solve it… This is of course a very selfish approach, but I think

[14] Debian is a popular open source community in which Linux software is packaged and built into a software distribution. The community consists of more than 600 programmers who "officially" maintain one or more software packages.

that most people work like that. They work on what they run into."[15] Thus, developers decide for themselves what they find interesting and what they like to work on: "the nice thing about open source is that you can make what you want to add."[16]

The fact that open source developers behave individualistic has three consequences. First, volunteers in the communities have no way of ensuring that others will do what they ask them to do. In other words, volunteers work independently from each other and no one can force them to do something they do not want to do. "Many free software developers don't work together. A company like Microsoft hires developers, put them in a room and make them work together. We don't because we do not have walls… We cannot make them do anything, we can only suggest."[17] This point coincides with the next statement: "In the discussion I can say "no" as hard as I like, but if he decides to change it, I cannot do anything to prevent it."[18]

Second, things only get done when someone wants to do it. "That is actually what the entire open source philosophy is all about. Things only get done if at least one person feels that they are important. This person will make sure that it works."[19] The general principle is always that: "decisions are made 'by those who are willing to do the work' and that in the long run, developers tend to support the best technical solution for a problem… (McCormick, 2003, p.31) Individual developers will "choose and select among" (McKelvey, 2001, p. 26) the open source programs that are available.

Third, volunteers in the community do not keep track of what others are doing: "we try and keep things simple. We like to spend more time getting stuff done instead of tracking everything."[20] The developers in the communities can be compared to ants in "a big ant colony: you do not know what someone else is doing."[21]

---

[15] Translated from Dutch, cited from an interview with a maintainer of various software modules in the Linux community.

[16] Translated from Dutch.

[17] Cited in an interview with the vice-president of the Free Software Foundation.

[18] Translated from an interview in Dutch with a translator of KDE software.

[19] Cited in an interview with one of the board members of the Apache Software Foundation.

[20] Cited from an interview with the project leader of the PostgreSQL community.

[21] Translated from Dutch, cited from an interview with a contributor in the KDE community.

The limited explanatory value of formal mechanisms

Most open source communities have two mechanisms that would theoretically have a role on a collective level, namely a voting system and project leadership structures. Theoretically, both mechanisms are important means to reach a collective choice or to resolve conflicts. However, both hardly interfere in many of the daily processes. This does not mean that they are not important or that they do not play a role in the organization of open source communities. However, they cannot explain much of the dynamics of the daily processes. These processes are outside the reach of these mechanisms and something else must be able to explain them.

The inability of project leaders to steer the community of developers is evidenced by a statement from a former project leader of a well-known open source community called Debian. In an interview he states: "Trying to lead Debian is liking trying to herd cats." Another example in which a formal mechanism is said to contribute to little is from a respondent who discusses the value of the voting system in the Apache community: He states:"The vote usually does not amount to anything much."[22] Another respondent argues how the voting system rather obstructs decision-making and encourages conflicts then resolves conflicts. He feels that it gives rise to a process in which "politics gets in and the original developer will get frustrated. He will want to add a patch and other people will vote negatively if they do not agree with the entire thing."[23] There are many more of such examples, which downplay the importance of formal mechanisms (see also Van Wendel de Joode, 2004a; Van Wendel de Joode et al., 2003; Van Wendel de Joode & Kemp, 2002)

Conflict resolution mechanisms in open source communities

It is outside the scope of this article to discuss the presence of each design principle in open source communities in detail. To indicate the relevance of CPR research and to discuss the way in which the design principles can shed a light on the organization of open source communities, this article will focus on just one of the eight design principles, namely *conflict resolution mechanisms*.

Open source communities are characterized by many conflicts (McCormick, 2003; Van Wendel de Joode, 2004). The next pages will provide a number of examples of conflicts. The question is: how are conflicts resolved in open source communities?

---

[22] The commit is used to refer to the act of developer who includes a piece of source code (patch) into the existing source code.

[23] Cited from an interview with two ASF members.

The role of modularity in conflicts

Most open source software is extremely *modular* (e.g. Benkler, 2002a; Bonaccorsi & Rossi, 2003; Kogut & Metiu, 2001; McKelvey, 2001). Consider the following statement from an active Linux programmer: "Of course, Linux software is very complex, but on the other hand of course it is not. The answer is to divide and conquer. When you have a complex piece of software, you cut it into ten pieces and if you manage to provide them with good interfaces then you only need to understand the separate pieces.[24]

Thus, modularity depends on two aspects, namely the *modules* themselves and the *interfaces* that connect the modules. Essentially, the modules are the building blocks of a software program, which perform separate tasks and which can act independently from each other. To have software that is divided into different modules has the big advantage that each module performs a limited set of tasks, which "individuals can tackle independently from other tasks." (Lerner & Tirole, 2002, p. 28). Modularity results in fewer conflicts for two reasons.

It increases *independence* between programmers and hence reduces the amount of conflicts. Programmers work on specific and relatively isolated parts of the software. Without modularity changes made in one part of the software would likely have the effect that "something else does not work anymore."[25] Therefore, programmers would have to make many decisions collectively and would be much more likely to frustrate each other's development efforts. Modularity decreases the dependency and hence the chance that a conflict occurs.

The fact that software is modular also enables the *localization and isolation of conflicts*. Consider any random conflict between two programmers about software that is not modular. In that case the conflict, even it were about a small part of the software, would very likely affect the entire software program and hence the entire community. Due to modularity, programmers have the option to break most conflicts down and attribute them to a small part of the software, i.e. to one module. Most conflicts can thus be localized to a specific module and can be discussed in isolation from the rest of the community.

---

[24] Translated from Dutch.

[25] Translated from an interview in Dutch. The interview was held with one of the two maintainers of the Lilypond software.

The role of parallel development lines in conflicts

The second mechanism is the creation of *multiple* or *parallel development lines*. This mechanism does not lead to the resolution of conflicts. It does, however, prevent conflicts from resulting in inertia. The Editor-in-Chief of Linux Journal explains: "It seems like there are often 2 different ways of tackling a problem and people try both." The creation of parallel development lines enables people to tackle the same problem in different ways and hence it stimulates creativity. It also provides a countervailing force against people who want to enforce their decision on others and it thus also stimulates the vitality of the community.

The basic premise is that parties with conflicting ideas, interests or values can start different lines of software development. By allowing them to start a new line the conflicting parties can actually write code the way they deem the most appropriate. These two or more lines will then compete, which is considered to be a good thing, "as it ensures that a project will continue to evolve and improve."[26]

In many cases it is so evident, which of the development lines is the better one that it becomes the de-facto standard in the community. In these situations the less popular lines "die" out because people simply stop investing time in its development (Egyedi & Van Wendel de Joode, 2003). The fact that projects are allowed to die ensures that the creation of many parallel lines of software development does not raise the level of redundancy and costs to extreme heights.

The stable and development version

There are different ways in which the mechanism of parallel software development lines is institutionalized in the communities. The most visible way is the institutionalization of a "stable" and "experimental" development line.

Non-technical users want a product they can download and install as easily as possible. Furthermore, they want the software to change as little as possible.[27] On the other hand, highly skilled and trained programmers want to work on exiting and challenging new tasks. This difference is bound to result in many conflicts. Most communities, however, have institutionalized a mechanism to deal with some of these conflicts, namely the presence of *a stable and an experimental software development line*. The founder of Linux International explains: "Linux consists of a production

[26] Cited from an interview with the chief editor of Linux Journal.

[27] Based on an interview with the creator and maintainer of the Python programming language.

version and development version. The even numbers are production versions; the odd numbers are for development. The development version is for trying out new things and testing. The even or production versions are the versions that are used in the distribution and indicate that the version will remain stable for a reasonable amount of time."

This mechanism is not only present in Linux; the presence of two "official" versions can also be found in communities like Debian, PostgreSQL, Apache and Python. The presence of two versions will ensure that conflicts come to surface less. Programmers and users that are less skilled and favor a stable software package will download a stable version of the software and are ensured that their interests are at least partly satisfied. Highly skilled programmers, on the other hand, will use and participate in the development of the experimental, i.e. the development version of the software. Now they are much more challenged to use their skills and test new ideas.

New heads in the CVS

Many programmers in free software communities use the Concurrent Versions System (CVS). A CVS is a client-server repository. It is an automated system that allows remote access to source code and according to the firm that develops and markets the CVS it enables multiple developers to work on the same version of the source code simultaneously.[28] Programmers commit source code to the CVS, which automatically sends an e-mail notification of the change to every member of the mailing list. Usually a number of programmers have an interest in the change and they are stimulated to download the newest version and test the modification. Regularly a programmer will dislike the modification. In that case there are a number of options open to him. One of the options could be to remove the latest addition from the CVS. This is likely to lead to much controversy, as others might have liked the modification or even deem the modification necessary. A conflict is present, if the programmers have a strong opinion about or interest in the source code and do not want to give in to each other arguments. So, how do programmers in the community deal with such a conflict?

In Apache the usual solution is to start, what is called a second *head*.[29] A head is the most recent version of a particular piece of source code in the CVS. Usually there is one head. Creating a second head, or in other words a second development line, allows people to develop and implement

[28] Based on information from the CVS home page: http://www.cvshome.org/ (last visited August 2002).

[29] Based on two interviews with board members of the ASF.

source code, the way they consider best. One of the members of the ASF board explains how the programmers in the community can monitor the development of both lines and judge for themselves what alternative they consider best and perhaps after a while "the community will decide to which alternative they agree."[30]

Here, the most important aspect of the creation of a second head is not the way in which the two lines are reintegrated again. Instead, the interesting part is how the second head does not solve the conflict but rather, diminishes the chance that it leads to inactivity or inertia. On the contrary, parties with conflicting ideas or interests, are actually stimulated to prove their point and thus to take an active role. Trying to prove that their alternative is the better one, programmers on both ends have to write new and better source code. Thus creating better software for the community as a whole.

A commercial development line

One of the sources for many conflicts is the difference between programmers who are paid by companies to develop open source software and "voluntary" developers. "One of the problems is that the individuals who develop on Apache that come from Sun have deadlines. That creates frustration, when they need to work together with people who do it for fun."[31] One of board members of Apache makes the same point when he says: "You could say there is a difference between a developer from for example IBM and a private developer who works at it for his hobby. The first will be working more structural towards deadlines and with a certain approach, whereas the latter is more creative and more risky in trying things and has generally a higher pay off risk ratio."

Obviously, these statements highlight a problem for companies that want to market a product based on open source software. They have to stick to deadlines, have to meet certain predefined performance levels and have to deliver a certain quality of service. Their current and potential customers will expect this from them. How can they keep these deadlines and at the same time diminish the chance of constant conflicts?

To overcome this problem, Covalent has come up with quite some clever solutions. Covalent has created its own Apache software development line, which enables it to develop the software independently from the community. In its version, Covalent can optimize and reconfigure

[30] Translated from an interview with one of the board members of the ASF.

[31] Cited from an interview with two programmers in the Apache community.

any part of the official Apache tree and ensure that its version best meets its customers' requirements. In this way, it also shields customers from the community. "The Apache Software Foundation may release four new versions in one week… our customers don't notice it though, because we isolate them completely from the ASF."[32] For Covalent, the advantage of disconnecting the development lines is that it no longer needs to pressure the community. "We just don't care if the ASF releases a new version or not."[33]

The dual development lines enable Covalent to disconnect its development efforts from the Apache development line. Covalent's efforts focus primarily on creating a product that better meets the wishes of its customers. Therefore, it runs a lot of tests and does a lot of bug-fixing. These bug fixes are added in Covalent's version immediately, but they are also sent back to the ASF, i.e. the Apache community. In most cases, the Apache programmers do add the fix in their version as well, thus profiting from Covalent's involvement. The commercial development line merely enables Covalent to keep a version in which mistakes are not tolerated, in which deadlines for new releases are met and in which the rise of conflicts is somewhat avoided.

The role of the exit option in conflicts

According to Hirschman (1970) the exit option is the underlying principle of all economic thought. It is the idea that exit is a way to signal the management of an organization that they are doing something wrong, at least in the eyes of its most important stakeholders, namely its customers and its members: "Some customers stop buying the firm's products or some members leave the organization: this is the exit option. As a result, revenues drop, membership declines, and management is impelled to search for ways and means to correct whatever faults have led to exit." (Hirschman, 1970, p. 4)

The exit option is different from the *voice option*, which is the direct expression of dissatisfaction (Hirschman, 1970, p. 4). Both the voice option and the exit option are ways for people to ventilate their dissatisfaction and disagreement. The difference is that the voice option is bound to lead to conflicts, as it is a direct confrontation of opposing ideas. Exit, however, does not lead to conflicts. One could say that it is a way of communicating dissatisfaction by doing.

[32] Cited from an interview with a board member from the ASF (translated from Dutch).

[33] Cited from an interview with a board member from the ASF (translated from Dutch).

The exit option and the voice option, Hirschman (1970) argues, are each other substitutes: "the presence of the exit option can sharply reduce the probability that the voice option will be taken up widely and effectively. Exit was shown to drive out voice, in other words…" (Hirschman, 1970, p. 76) Given the fact that the voice option will lead to conflicts, this statement implies that the exit option will diminish the number of conflicts, because it is an effective way to substitute voice. When people have the possibility to exit, they will use their voice less and will thus create fewer conflicts. To identify the presence of an exit option will result in the conclusion that the voice option is used less and therefore there is less conflict. It also implies that the presence of an exit option is a way to resolve conflicts.

The search for private success

Guido van Rossum, the project leader of the Python community, argues how certain developers feel the need to exit the Python community: "Sometimes developers have the idea that they can make a tool that can work with Python, but that there might be no place for this in Python. Sometimes they start their own project in which they can specify their solution or idea, like Phyrex or Phycom."

Psycho is also an example of such a project. A Swiss PhD student named Armin Rigo developed Psycho. His goal is to make the Python programming language faster, but currently the solution is not included in the Python software. Psycho, however, is becoming more mature and has proven to really make the language faster. According to Python's project leader there is a good chance that in the near future the tool will be included in an official Python release, as it has proven its value.[34]

The development of Psycho is a good example of the exit option at work in a free software community. Armin Rigo wanted to develop software that would make Python, which is a higher-level programming language, faster than lower level languages. Rigo knew, however, that this idea contradicts popular belief, which presumes that a higher-level programming language is always slower than a lower level language.[35] Of course he could have stayed in the community and have tried to develop the tool there. This, however, was likely to lead to much resistance and hence a lot of conflict, as people did not believe this to be possible.

---

[34] Based on an interview with the creator and current project leader of the Python programming language.

[35] Based on an interview with Armin Rigo.

The decision to exit the community, to seek private success and thus to develop the tool outside the Python community has enabled Armin Rigo to develop Psycho without running into these conflicts. In other words, his decision to exit has prevented the occurrence of many potential conflicts.

The fork as exit option

The most dramatic exit option is undoubtedly the *fork*. According to the jargon file, a dictionary for free software programmers, a fork is "What occurs when two (or more) versions of a software package's source code are being developed in parallel which once shared a common code base, and these multiple versions of the source code have irreconcilable differences between them."[36] Most forks start with a conflict: with a difference in opinion, value or interest between two or more parties. Then one of the parties decides to take the source code and start a new community based on that same source code. After a while the differences between the two communities are irreconcilable and the fork is complete.

Much research considers forks to be negative. Narduzzo and Rossi (Narduzzo & Rossi, 2003) for example state: 'These forks are an expression of a coordination failure…' (p. 29) Kuwabaru (2000) has a similar stance when he writes: 'Forking… and other behaviors become taboos.' (Kuwabara, 2000, p. 48) In other words, both authors argue that forking is something bad, something that should be avoided.

However, it is argued here that the fork should be viewed as a way to prevent or solve conflicts. Viewed in this perspective forks are not inherently bad, but can have a positive influence on the continuity of open source software development. The negative aspect of forking is that it can lead to a destruction of value. It results in the creation of two communities, which initially will have little differences between them. To have two communities spending resources on more or less the same source code is a waste of those resources. It is hence understandable that programmers stress that one should as long as possible sustain from using this mechanism. They also recognize, however, that it is sometimes impossible to continue working together. In that situation it is justified and perhaps even recommendable to fork a project. A former project leader of Debian states: "It is

[36] Taken from the Internet: http://jargon.watson-net.com/jargon.asp?w=fork (last visited on April 7th, 2003)

essential that you can decide to leave if you do not agree. There is a rule, that you should not do it if you do not think that it is strictly necessary. But sometimes it is necessary!"[37]

Conclusion

This article has introduced open source communities. Furthermore, it has argued why the eight design principles from research on community managed common pool resources are promising to adopt as a framework. Applying the framework of CPR on open source communities will contribute to:

1) An increased understanding of the organizations of open source communities.

Applying the framework of research on common pool resources can provide an insight in the issues that surround the organization of open source communities. Decision-makers from companies and government agencies will benefit from this insight, as they can better understand the threats that face open source communities and can better understand what their potential role and activities are to contribute to the continuity of open source software development and maintenance. Furthermore, it can make decision-makers and open source programmers understand that certain perceived negative aspects can have a positive influence on the continuity of open source communities.

This article provided a number of mechanisms that play a vital role in the way in which open source communities deal with conflicts.. They were: forking, modularity, commercial development lines, a stable and a development version and a new head. Many of these mechanisms might appear to be inefficient and even irrational. This article, however, demonstrates that using the CPR perspective is likely to shed a different light on these mechanisms and demonstrates how these mechanisms do contribute to the continuity of the communities. The question of course is: do the benefits outbalance the costs, as they do result in an increasing level of inefficiency and redundancy. More research is needed to understand what exactly the trade-off is and when it is appropriate to actually start or even institutionalize a second development line and when not.

2) An increased understanding of the way in which CPR research can be applied in an ICT environment.

[37] Translated from Dutch.

It was argued that more and more people adopt the discourse from research on common pool resource on ICT networks and ICT goods. So far, however, this has frequently stopped at the level of making analogies. Hess and Ostrom (2001) try to take the discussion one step further. They conclude that the ideas from CPR research do apply to certain ICT goods. However, they feel that the design principles should be different. Perhaps this is true.

Some first results from this research suggest that the design principles might be more appropriate then Hess and Ostrom argue them to be. Applying the design principles on a concrete case can shed light on the applicability of the design principles in an ICT environment and can lead to valuable theoretical insights in the factors that influence the continuity of common pool resources and can thus provide a valuable contribution to enrich the CPR framework.

References

Anderies, J. M., Janssen, M. A., & Ostrom, E. (2003). *Design Principles for Robustness of Institutions in Social-Ecological Systems*. Paper presented at the Joining the Northern Commons: Lessons for the World, Lessons from the World, Anchorage.

Axelrod, R., & Cohen, M. D. (1999). *Harnessing Complexity: Organizational Implications of a Scientific Frontier*. New York: Free Press.

Bekkers, V. J. J. M. (2000). *Voorbij de virtuele organisatie? Over de bestuurskundige betekenis van virtuele variëteit, contingentie en parallel organiseren*. 's Gravenhage: Elsevier bedrijfsinformatie.

Benkler, Y. (2002a). Coase's Penguin, or, Linux and the Nature of the Firm. *Yale Law Journal, 4*(3).

Benkler, Y. (2002b). Intellectual Property and the Organization of Information Production. *International Review of Law and Economics, 22*(1), 81-107.

Berkes, F. (2000). *Cross-Scale Institutional Linkages: Perspectives from the Bottom Up*. Paper presented at the IASCP 2000 Conference, Indiana University.

Bernbom, G. (2000). *Analyzing the Internet as a Common Pool Resource: The problem of Network Congestion*. Paper presented at the International Association for the Study of Common Property 2000, Bloomington, Indiana, USA.

Bessen, J. (2002). What good is free software? In R. W. Hahn (Ed.), *Government Policy toward Open Source Software* (pp. 12-33). Washington: AEI-Brookings Joint Center for Regulatory Studies.

Blumer, H. (1954). What is strong with social theory. *American Sociological Review, 19*, 3-10.

Bollier, D. (2001). *Public Assets, Private Profits; Reclaiming the American Commons in an age of Market Enclosure*. Washington: New America Foundation.

Bonaccorsi, A., & Rossi, C. (2003). Why Open Source Software can succeed. *Research Policy, 32*(7), 1243-1258.

Boyle, J. (2003). The second enclosure movement and the construction of the public domain. *Law and contemporary problems, 66*(1/2), 33-74.

Bruns, B. (2000). *Nanotechnology and the Commons: Implications of Open Source Abundance in Millennial Quasi-Commons*. Paper presented at the Constituting the commons, Bloomington, Indiana.

Buck, S. J. (1998). *The Global Commons, An Introduction*. London: Earthscan Publications.

Cowan, R., & Harison, E. (2001). Protecting the digital endeavor: prospects for intellectual property rights in the information society. *AWT-achtergrondstudie nr. 22*.

Davies, J. (2001). *Traditional CPRs, New Institutions: Native Title Management Committees and the State-Wide Native Title Congress in South Australia*. Paper presented at the Tradition and Globalisation: Critical Issues for the Accommodation of CPRs in the Pacific Region, the Inaugural Pacific Regional Meeting of the International Association for the Study of Common Property, Brisbane, Australia.

Edwards, K. (2001). *Epistemic communities, situated learning and open source software development*. Paper presented at the 'Epistemic Cultures and the Practice of Interdisciplinarity' Workshop at NTNU, Trondheim.

Egyedi, T. M., & Van Wendel de Joode, R. (2003). *Standards and coordination in open source software*. Paper presented at the 3rd IEEE Conference on Standardization and Innovation in Information Technology, Delft.

Egyedi, T. M., & Van Wendel de Joode, R. (2004). Standardization and other coordination mechanisms in open source software. *Journal of IT Standards & Standardization Research, 2*(2).

Fielding, R. T. (1999). Shared Leadership in the apache Project. *Communications of the Association for Computing, 42*(4), 42/43.

Fontana, A., & Frey, J. H. (1994). Interviewing: the art of science. In N. K. Denzin & Y. S. Lincoln (Eds.), *Handbook of Qualitative Research*. Thousand Oaks: Sage Publications.

Garzarelli, G. (Forthcoming). Open Source Software and the Economics of Organization. In J. Birner & P. Garrouste (Eds.) (Third draft, Version 3.06; January 13, 2002 ed.): Routledge.

Hardin, G. (1968). The Tragedy of the Commons. *Science, 162*, 1243-1248.

Hertel, G., Niedner, S., & Herrmann, S. (2003). Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Research Policy, 32*, 159-1177.

Hess, C., & Ostrom, E. (2001). *Artifacts, Facilities, And Content: Information as a Common Pool Resource.* Paper presented at the Conference on the Public Domain, Duke Law School.

Hirschman, A. O. (1970). *Exit, Voice, and Loyalty: responses to decline in firms, organizations, and states.* Cambridge, Massachusetts and London: Harvard University Press.

Hunter, D. (2003). Cyberspace as Place and the Tragedy of the Digital Anticommons. *California Law Review, 91*, 439-519.

Iannacci, F. (2003). The Linux Managing Model. *First Monday. Peer reviewed journal on the Internet, 8*(12).

Kogut, B., & Metiu, A. (2001). Open-Source software development and distributed innovation. *Oxford Review of Economic Policy, 17*(2), 248-264.

Kollock, P., & Smith, M. A. (1996). Managing the Virtual Commons: Cooperation and Conflict in Computer Communities. In S. Herring (Ed.), *Computer-Mediated Communications: Linguistic, Social, and Cross-Cultural Perspectives* (pp. 109-128). Amsterdam: John Benjamins.

Kuwabara, K. (2000). Linux: A Bazaar at the Edge of Chaos. *First Monday. Peer reviewed journal on the Internet, 5*(3).

Lakhani, K., & Von Hippel, E. (2003). How Open Source Software Works: "Free" User-to-User Assistance. *Research Policy, 32* (Special issue on open source), 922-943.

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

Madey, G., Freeh, V., & Tynan, R. (2002). *Understanding OSS as a Self-Organizing Process.* Paper presented at the Meeting Challenges and Surving Success: 2nd Workshop on Open Source Software Engineering, Orlando.

Markus, M. L., Manville, B., & Agres, C. E. (2000). What Makes a Virtual Organization Work? *Sloan Management Review, 42*(1), 13-26.

Mauss, M. (1990). *The gift: the form and reason for exchange in archaic societies* (Vol. Translation of Essai sur le Don (1950) Presses Universitaires de France). London: W. W. Norton.

McCormick, C. (2003). *"The Big Project That Never Ends": Role and Task Negotiation Within an Emerging Occupational Community.* Unpublished Recent summary of findings, as part of PhD research, University of Albany.

McGowan, D. (2001). Legal Implications of Open-Source Software. *University of Illinois Review, 241*(1), 241-304.

McKelvey, M. (2001). *Internet Entrepeneurship: Linux and the dynamics of open source software* (Discussion Paper). Manchester: Centre for Research on Innovation and Competition, The University of Manchester.

Moon, J. Y., & Sproull, L. (2000). Essence of Distributed Work: The Case of the Linux Kernel. *First Monday. Peer reviewed journal on the Internet, 5*(11).

Muchapondwa, E. (2002). *Sustainable Commercialised Use of Wildlife as a Strategy for Rural Poverty Reduction: The Case of 'Campfire' in Zimbabwe*. Paper presented at the The Commons in an Age of Globalisation, the Ninth Conference of the International Association for the Study of Common Property, Victoria Falls, Zimbabwe.

Narduzzo, A., & Rossi, A. (2003). Modularity in Action: GNU/Linux and Free/Open Source Software Development Model Unleashed. *Quaderno DISA n. 78*.

Olson, M. (1965). *The logic of collective action; public goods and the theory of groups*. Cambridge: Harvard University Press.

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

Ostrom, E. (1990). Governing the Commons; The Evolution of Institutions for Collective Action. In J. E. Alt & D. C. North (Eds.), *The political economy of institutions and decisions* (fourth ed.). Cambridge: Cambridge University Press.

Ostrom, E. (1998). A behavioral approach to the rational choice theory of collective action. *The American Political Science Review, 92*(1), 1-22.

Ostrom, E. (1999). Coping With Tragedies of the Commons. *Annual Review Political Science, 2*, 493-535.

Ostrom, E. (2000). Collective Action and the Evolution of Social Norms. *Journal of Economic Perspectives, 14*(3), 137-158.

Raymond, E. S. (2000). Homesteading the Noosphere. *First Monday. Peer reviewed journal on the Internet, 4*(10).

Raymond, E. S. (2001). *The Cathedral and the Bazaar*: O'Reilly.

Sarker, A., & Itoh, T. (2001). Design principles in long-enduring institutions of Japanese irrigation common-pool resources. *Agricultural Water Management, 48*(2), 89-102.

Sekher, M. (2001). Organized participatory management: insights from community forestry practices in India. *Forest Policy and Economics, 3*, 137-154.

Sharma, S., Sugumaran, V., & Rajagopalan, B. (2002). A framework for creating hybrid-open source software communities. *Information systems Journal, 12*(1), 7-25.

Smith, A. D. (1999). Problems of conflict management in virtual communities. In M. A. Smith & P. Kollock (Eds.), *Communities in Cyberspace* (pp. 134-163). London: Routledge.

Spencer, L., Ritchie, J., & O'Connor, W. (2003). Analysis: Practices, Principles and Processes. In J. Ritchie & J. Lewis (Eds.), *Qualitative research practice: A guide for social science students and researchers* (pp. 199-218). Londen: Sage Publications.

Tang, S. Y. (1992). *Institutions and Collective Action: Self-Governance in Irrigation*. San Francisco: Institute for Contemporary Studies.

Torvalds, L., & Diamond, D. (2001). *Gewoon voor de fun* (C. Jongeneel, Trans.). Uithoorn: Karakter Uitgevers.

Travers, M. (2001). *Qualitative Research Trough Case Studies*. London: Thousands Oaks.

Tucker, C. (1998). *Evaluating a Common Property Institution: Design Principles and Forest Management in a Honduran Community*. Paper presented at the "Crossing Boundaries", the seventh annual conference of the International Association for the Study of Common Property, Vancouver, British Columbia, Canada.

Van Wendel de Joode, R. (2004a). Conflicts in open source communities. *Electronic Markets, 14*(2).

Van Wendel de Joode, R. (2004b). *Continuity of the commons in open source communities*. Paper presented at the SSCCII-2004 Conference, symposium of Santa Caterina on Challanges in the Internet and Interdisciplinary Research (IEEE cosponsored conference), Amalfi, Italy.

Van Wendel de Joode, R., De Bruijn, J. A., & Van Eeten, M. J. G. (2003). *Protecting the Virtual Commons; Self-organizing open source communities and innovative intellectual property regimes*. The Hague: T.M.C. Asser Press.

Van Wendel de Joode, R., & Kemp, J. (2002). The Strategy Finding Task within Collaborative Networks, based on an exemplary Case of the Linux Community. In L. M. Camarinho-Matos (Ed.), *Collaborative Business Ecosystems and Virtual Enterprises* (pp. 517-527): Kluwer Academic Publishers.

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

Wayner, P. (2000). *FREE FOR ALL: How Linux and the Free Software Movement Undercut the High-Tech Titans*. New York: HarperBusiness.

Weber, S. (2000). *The Political Economy of Open Source Software* (BRIE Working Paper 140, E-conomy Project Working Paper 15, available on the Internet: http://brie.berkeley.edu/pubs/pubs/wp/wp140.pdf).

Yin, R. K. (1989). *Case study research: design and methods* (second ed.). Newbury Park: Sage Publications.