Organizing for continuity in a digital information commons: the institutions in open source communities

Ruben van Wendel de Joode
Faculty of Technology, Policy and Management
Delft, University of Technology
PO Box 5015, 2600 GA, Delft
The Netherlands
rubenw@tbm.tudelft.nl
(t) +31(0)152781105
(f) +31(0)152786233

Abstract

Open source communities are groups of sometimes hundreds if not thousands of individuals with differing interests, backgrounds and motives. Collectively, these individuals develop software and perform related tasks like writing documentation and answering user questions. In the communities, the source code, which is the human-readable part of the software, is freely available, downloadable and modifiable. This characteristic of open source communities is the main reason why many researchers have claimed that open source software and the corresponding source code are in a commons.

The main goal in this paper is to analyze and describe how open source communities are organized and how they sustain themselves. Both the analysis and the description are based on and structured along the eight design principles as first described by Elinor Ostrom (1990).

One of the important findings in this paper is that we cannot understand the organization of open source communities based on formal institutions alone. It is shown how a vast array of mechanisms structure and influence the behavior of individuals. Combined the mechanisms and institutions "solve" the issues addressed in each of the design principles.

Introduction

Open source software (OSS) communities are internet-based networks of individuals who collaboratively write software and perform many related activities like writing documentation and solving user problems. Some OSS communities have attracted hundreds of individuals who have different interests, backgrounds and motives (Ghosh 2005) and a large percentage of these individuals are volunteers (Hertel et al 2003). In the communities the source code, which is the human-readable part of software, is available, downloadable and modifiable. As such the communities enable almost anyone from anywhere in the world to contribute to the improvement of the software (Von Krogh & Von Hippel 2003). The software and the corresponding source code are said to be in a *commons*. (Benkler 2002a, Bollier 2001, Bruns 2000, Kogut & Metiu 2001, McGowan 2001) To support the distributed development process individuals in the communities have created an infrastructure of mailing lists, versioning systems and bug report systems.

OSS communities are scientifically interesting because they appear to lack many mechanisms that are frequently claimed to be essential to ensure coordination. For instance, the communities lack labor contracts or more general contractual relationships that tell participants what they should do and how (Franck & Jungwirth 2003). Participants decide what they want to work on and how they want to do it. "Work is not assigned; people undertake the work they choose to undertake" (Mockus et al 2002, p. 310). Furthermore, open source communities lack clearly defined organizational boundaries (Fielding 1999, Raymond 1999). "Membership in the community is fluid; current members can leave the community and new members can also join at any time" (Sharma et al 2002, p. 10). Absence of these mechanisms results in a puzzle, namely, 'How are the communities organized?'

This puzzle is further complicated by the internal and external pressures that face the communities. The external pressure emanates from the fact that the communities are intertwined with the software market. The communities are confronted with intellectual property rights which can be used to appropriate software from the commons (Benkler 2002b, Bollier 2001, Bollier 2002, Boyle 2003). Patents and copyrights, but also the commercialization of open source software, poses external pressures to OSS communities and their continuity (Vemuri & Bertone 2004, Van Wendel de Joode et al 2003). The internal pressures, on the other hand, are closely related to the organization of OSS communities, and includes, for example, social dilemmas like free riding (Von Hippel & Von Krogh 2003, McGowan 2001) and cascading conflicts.

Despite the above, individuals in the communities are able to collectively develop software that is highly complex and, according to some definitions, is successful, as evidenced by the increasing number of organizations and governments that are turning to open source software to facilitate their critical business processes.[1] Furthermore, programs like Sendmail and Apache dominate their respective segments of the software market. Also open source software like Apache and Linux has proven to be of comparable quality to proprietarily developed software. [2]

---

[1] One example is the *New York Stock Exchange*: http://www.it-director.com/article.php?articleid=2125 (November 2003) and in May 2003 the city council of Munich decided to switch 14,000 desktops away from Microsoft to open source software (http://www.wired.com/news/infostructure/0,1377,62236,00.html, March 2004).

[2] From the Internet: http://www.infoworld.com/article/03/07/01/HNreasoning_1.html (March 2004) and http://www.reasoning.com/news/pr/02_11_03.html (July 8, 2003).

The fact that OSS communities face different types of pressures and yet the software has been able to gain a large share of the market gives rise to our research question:

How are open source communities organized and how do they sustain themselves?

Two terms in this question require explanation. *To organize* implies that having motivated individuals is not enough. Much research on OSS communities has concentrated on the question of why individuals are motivated to participate (e.g. Hars & Ou 2002, Hertel et al 2003, Lakhani & Von Hippel 2003). Clearly, however, an organization is more than a collection of motivated individuals. An 'organization' implies the presence of coordination, which calls attention to processes like gathering and mobilizing resources, allocating resources, and negotiating and structuring responsibilities and activities among individuals. *To sustain* refers to the fact that the existence or continuation of OSS communities over time is far from obvious or logical. By and large, research on OSS communities appears to take their presence as a given, as if it needs no further inquiry. Such research concentrates on many relevant questions, but not on the question how a collective of individuals survives over a longer period of time.[3] How do the communities protect their continuity and that of the software commons in light of internal and external pressures? This research aims to fill both gaps in current research on open source communities.

The research framework

Two explanations from state-of-the-art research on open source

State-of-the-art research on the organization of OSS communities can be roughly divided into two groups. The first group is made up of researchers who argue that OSS communities are self-organizing systems (Axelrod & Cohen 1999, Bekkers 2000, Kuwabara 2000). The communities are not created according to a grand design and centralized control is absent. Lanzara and Morner write, "Open-source software projects are made of heterogeneous components that keep a dynamic balance with one another. The balance does not come from *ex ante* or centrally planned design, but rather emerges out of unplanned, decentral interaction" (2003, p. 11). Coordination in a self-organizing system is the result of the actions of agents and their interactions with other agents and local conditions. Research on self-organizing systems typically uses rules to represent and describe agents' actions (Bonabeau et al 1999, Holland 1995, Kelly 1994, Resnick 1994). Such rules are behavioral, as they are said to describe the behavior of agents.

The second group is comprised of researchers who analyze specific elements of the organization of OSS communities. This type of research has identified a great number of collective mechanisms and institutions. Institutions are defined here as the factors and forces that constrain or support the behavior of individuals *and* that give rise to regularities in the patterns of human behavior (Crawford & Ostrom 1995). Examples of the institutions that can be found in open source communities are project leadership, foundations and open source licenses (e.g. Bonaccorsi & Rossi 2003, Hann et al 2002, McGowan 2001, O'Mahony

[3] One exception is an article by O'Mahony SC. 2003. Guarding the Commons: How Community Managed Software Projects Protect Their Work. *Research Policy* 32: 1179-98. She explicitly addresses how OSS communities protect themselves against external pressures. She then focuses on the boundaries constructed in OSS communities.

2003). The implicit claim in this line of research is that institutions explain how the communities are organized, as they determine how individuals behave.

The two groups of researchers forward explanations that are contradictory in important respects. One group claims that the communities are collections of interacting agents. Agents make their own informed choices based on local information and any form of centralized control is absent. This explanation appears to be in conflict with the second explanation, which is based on institutions. Quite a number of researchers have identified and discussed the role of institutions in OSS communities. The claim is that these institutions influence and determine the behavior of individuals. Therefore, the institutions explain how (parts of) the communities are organized. OSS communities should not be thought of as collectives of interacting agents because such an explanation is overly simplistic and metaphorical (Kogut & Metiu 2001, Weber 2004).

Reconciling differences: lessons from community-managed common pool resources

The framework adopted in this research to analyze the organization of OSS communities is based on the lessons from research on community-managed common pool resources. One might be tempted to say that these lessons are not appropriate to study OSS communities, since software is not a common pool resource. However, there are a number of reasons that justify such choice (see also Kollock 1996, O'Mahony 2003). The first and foremost reason for adopting the lessons from research on community-managed common pool resources is that it incorporates explanations based on self-organization *and* institutions. They are treated as complementary. The second reason is that much of the research on community-managed common pool resources is developed and tested through empirical research. A third reason is that research on common pool resources and the resulting framework are widely accepted and have been tested in many different settings and by a variety of researchers. Finally, open source software and the communities do face a threat of appropriation, which is enabled by the presence of patents, and therefore their future availability is to some degree threatened (O'Mahony 2003, Van Wendel de Joode 2004b).

Table 1 - Design principles of long-enduring communities

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

Ostrom (1990, 1999) and many of her colleagues, most of whom either work at or are associated with the Workshop in Political Theory and Policy Analysis at Indiana University, have performed extensive research on self-organizing communities in which common pool resources are managed. One important outcome of this research is the identification of eight *design principles* (see table 1). These principles are said to explain why and how communities are able to organize and sustain themselves, and the resource they manage (Ostrom 1990). The design principles are basically generalized descriptions of institutions that communities

craft to organize and sustain themselves over longer periods of time (Ostrom 1993). Individuals in the communities are affected by the institutions and change their behavior because they are aware of the rules and because they expect nonconformance to be monitored and sanctioned.

The design principles were used in this research in a slightly revised form as a heuristic to understand the organization of OSS communities. The communities were analyzed and observations structured according to the design principles. The underlying assumption is that the presence of institutions as described by the design principles would provide a conceptual explanation of how OSS communities are able to organize and sustain themselves. In total 60 interviews were held with participants in OSS communities.

Boundaries

The first design principle is the presence of clearly defined boundaries. Boundaries in communities are needed to create a feeling of belonging, to determine who is outside and inside and to decide who has a right to the fruits of the efforts of the community members. Research on common pool resources stresses that not only must organizational boundaries be created; boundaries delineating the resource are equally important (e.g. Ostrom 1990).

Boundaries are also needed in OSS communities, as the resource is susceptible to depletion. Yet, what is striking is that the communities in general lack a boundary to limit the size of their membership. In other words, organizational boundaries tend to be absent. The boundaries present in OSS communities are primarily intended to demarcate the boundaries of the source code.

The most important boundary in OSS communities is undoubtedly the open source license (see for instance Benkler 2002b, Dalle & Jullien 2003, Lerner & Tirole 2002a, McGowan 2001). The licenses protect open source software against appropriation (O'Mahony 2003).[4] A number of observations can be made concerning the boundaries. First, each individual or company can always create a new license, which has resulted in an wide variety of open source licenses. The Open Source Initiative (OSI), which is a foundation established to promote the concept of open source, lists 48 licenses that comply with its definition of open source.[5] The differences between most of these licenses are small, and yet individuals and companies have taken the time and effort to create these variations to suit their specific needs.

The licenses are dynamic and responsive to emerging threats and challenges. The environment of open source software development is not static. Technology is constantly changing, and companies and individuals are constantly exploring the limits of what the licenses allow them to do with the software. The creators and maintainers of the licenses must anticipate and respond to these changes and new challenges. An example of a license that needed change is the GPL. "When writing a license, one can only see a limited number of years ahead. Currently the GPL 2.0 has trouble addressing certain issues."[6] The GPL version 3.0 or the Affero GPL was written to address some of these issues.

---

[4] Next to the licenses communities have other boundaries to protect them against external pressures. These include, for instance, trademarks and foundations. Ibid.. This section, however, is limited to open source licenses and foundations, as they are believed to be the most important boundaries in open source communities.

[5] Measured September 10, 2003.

[6] From an interview with the vice-president of the FSF.

There is convergence in the selection of open source licenses. Most OSS communities, as if it were almost a collective decision, are governed by a limited number of 'favorite' licenses. For instance, the GPL is by far the most used license.[7] Furthermore, there are many relatively obscure and unknown licenses that are largely ignored and which are used by only a very small number of communities.

Next to open source licenses, many communities have erected a legally constituted foundation to protect the individual participants (O'Mahony 2003). These foundations provide a powerful protection mechanism against the appropriation of software; they create a nonprofit tax status and they enable legal representation of the community and individual contributors. But, they are also non-intrusive, as they hardly interfere with the actual processes within the communities. They also leave ample space for individual decisions, flexibility and change.

A third boundary in OSS communities consists of the large set of mailing lists that participants actively use. These mailing lists serve as 'boundary spanners'; they educate about the licenses, they filter relevant external information to the participants and they provide a means to organize pressure on people or entities that infringe on the licenses.

Of the three boundaries, the open source licenses are the most important. They define what users are allowed to do with the software and what not. The licenses can be said to be embedded in a system of foundations and mailing lists. The latter two educate about the licenses and provide a vehicle to enforce them when needed. One could say that the foundations are formally erected institutions to, among other things, legally enforce the licenses, and that the individuals on the mailing lists informally punish organizations and individuals who try to exploit loopholes in the licenses.

Appropriation and provision rules

To ensure the continuity of the source code, theory would suggest that the communities need both appropriation and provision rules. Appropriation rules aim to determine how much the members of a community are allowed to consume. Provision rules regulate how and when members are supposed to contribute their time and effort to the development and maintenance of the resource.

OSS communities have no appropriation rules other than the licenses mentioned in the previous section. This lack of additional appropriation rules is partly explained by the fact that many types of software usage are non-subtractive. This means that use does not reduce the amount available to others. The other part of the explanation is that there are boundaries that restrict the types of usage that do result in a reduction of software available to others. For this reason, attention is best focused on the way in which provision is organized. It is important to note that the mechanisms identified and described apply only given the presence of a sufficient number of individuals motivated to contribute. Thus, only in the larger communities that attract many participants will these mechanisms be sufficient to ensure the provision of software.

First, software development and maintenance in OSS communities is characterized by an apparent lack of body or platform which actually defines or delegates activities among the participants. For instance, many communities have project leaders and/or maintainers. However, these positions carry little real authority and lack the ability to assign tasks to

[7] According to the statistics on the Freshmeat website, 65 percent of the projects listed use the GPL. From the website software.freshmeat.net/stats/ (May 21, 2003).

individual participants (see also Markus et al 2000, p. 21). Furthermore, there is no body that actually monitors progress on tasks and activities. Development of software in the communities is instead based on individual choice and a 'culture of doing', which is explained in some more detail under the heading of *conflict resolution mechanisms*.

Second, elegance[8] and modularity[9] lower the need to coordinate activities of participants. Elegance reduces the investment required for participants to understand software written by others, as the software itself is able to explain "what it is doing while you are reading it."[10] Obviously, different people will interpret software differently. However, elegance makes the software easier to understand. Furthermore, elegance eases the implementation of changes. The big advantage of modular software is that each module performs a limited set of tasks, which "individuals can tackle independently from other tasks" (Lerner & Tirole 2002b, p. 28). Therefore, for modular software the need to formally divide and coordinate activities among participants is reduced[11] and the testing of new patches is simplified (Torvalds 1999).

Third, many participants voluntarily use a great number of primarily technical tools and mechanisms to structure their activities and enable others to find and understand software. The mechanisms are (i) a concurrent versions system, which many communities use to structure the actual development process (Bauer & Pizka 2003, Shaikh & Cornford 2003); (ii) mailing lists, which allow one-to-many communication to discuss ideas and exchange knowledge (Bauer & Pizka 2003); (iii) bug-tracking systems, which enable participants to report bugs that can be addressed and solved by others; (iv) manuals and coding style guides,[12] which prescribe how software should be written; (v) to-do lists, which provide an inventory of functionalities that are wanted and are open for work; (vi) the 'orphanage,' as in the Debian community, where software that is in need of a new maintainer is stored;[13] (vii) text added in the source code to signal and communicate source code characteristics; (viii) names attached to improvements, which provides a means for others to contact the author; and (ix) small and incremental patches, which make changes relatively easy to understand and which ease improvements or reversals to these changes.

---

[8] Elegant source code is that which is structured and reductive. It is a term used to indicate that the software works and, at the same time, is a notion of beauty. "So from the standpoint of a small group of engineers, you're striving for something that's structured and lovely in its structuredness." Cited from an interview with Ellen Ullman by Scott Rosenberg in 1997. The interview was published on the Internet: http://archive.salon.com/21st/feature/1997/10/09interview.html (August 2002).

[9] Modularity is based on the idea of divide and conquer, see: Dafermos GN. 2001. Management and virtual decentralised Networks: The Linux Project. *First Monday. Peer reviewed journal on the Internet* 6: downloaded from the Internet: http://www.firstmonday.dk/issues/issue6_11/dafermos/ (December 2004) Modular software is divided into smaller pieces, building blocks, which together create a working software program.

[10] From an interview with the editor in chief of the Linux journal.

[11] Agreement is required on what the modules are and how the interfaces should be defined. This could result in much conflict and disagreement. However, for this too the answer appears to be in 'doing,' as evidenced in many communities analyzed in this research. For instance, in the Apache community one of the developers sat down and decided to rewrite the HTTP server and made the design modular. Others accepted and adopted the changes. The principle of doing is explained more elaborately elsewhere in this book.

[12] See www.apache.org/dev/styleguide.html (May, 2003) and www.linuxjournal.com/article.php?sid=5780 (May, 2003).

[13] The website can be found at http://www.debian.org/devel/wnpp/work_needing (April 8, 2003).

Conflict resolution mechanisms

Conflicts can have both negative and positive effects on organizations. Some of the advantages are said to be a higher level of productivity, a rise in creativity and an increase in vitality (Jehn 1995, Rosenthal 1988). However, conflicts can also have negative effects. They can become destructive and result in inactivity (e.g. Jehn & Mannix 2001). Communities need some way to manage and resolve conflict, to prevent conflict from threatening the continuity of the community (Ostrom 1990).

OSS communities have a high potential for conflict. The literature describes at least two sources of conflict: *interdependency* (Dipboye et al 1994, Jehn 1995) and *diversity* (Deutsch 1973, Gefu & Kolawole 2002). In OSS communities the participants are mutually dependent. Participants cannot develop and maintain the software alone; they need contributions from others, as well as maintain compatibility with other pieces of software. There is also a high level of variety among open source developers (Markus et al 2000) and they differ widely in what they want the software to do. Thus, many decisions may invoke conflicting positions. For instance, consider a situation in which two participants have created a different solution to the same problem. Both spent time to create their solution. They will be tempted to defend their solution and explain to others why their solution is the better one.

The high potential for conflicts is tempered through the software's modular design. Modularity increases the independence of participants. It ensures that changes made in one part of the software are less likely to have the effect that "something else does not work anymore."[14] Modularity also enables conflicts to be localized and isolated. Indeed, programmers can break down most conflicts into smaller parts and attribute them to a part of the software, that is, to one module.

In the communities, conflicts are plentiful, but their negative consequences are relatively minor. A number of reasons can be given for this. First, conflicts are transformed into a competition between two or more alternatives. Thus, conflicts are not so much resolved, but translated into parallel development lines. In this way, conflicts do not disrupt the activities of individuals in the communities. Second, participants in the communities have easy access to the exit option. The exit option dampens the emergence of conflicts (Hirschman 1970), because people can vote with their feet. If participants disagree strongly, they can exit the community. An extreme example of the exit option is the *fork*. At the heart of a fork are two parties that disagree, for instance, about the direction of a software development project. One of the parties could decide to exit the community, but still continue the development of software in the way they consider best. At a certain point in time the two programs are likely to develop irreconcilable differences, at which point the fork is complete.

The underlying principle of both the parallel development lines and the exit option is the 'culture of doing.' Quite a number of respondents argued that they frequently ignore conflicts on mailing lists and in other community forums. They just do what they think is right. "So if I think something should be changed then I do it myself, particularly when the group disagrees."[15] The basic principle is that if participants want to convince others in the community that they are right, they have to provide the proof. They do this by producing the source code and demonstrating that their solution is better. In other words, participants must put their code where their mouth is. They quickly lose interest and ignore conflicts surrounding hypothetical solutions.

---

[14] From an interview with one of the two maintainers in the Lilypond community.

[15] From an interview with a member of the KDE community.

Collective choice

A collective choice is a particular type of choice, namely, one that binds and restricts the individuals in a collective. The "decisions made in collective-choice situations *directly* affect operational situations" (Ostrom 1990, p. 192). The most commonly mechanism to reach a collective choice, as described in literature on collective choice, is through voting systems (Arrow 1951, Walker et al 2000).

Voting systems are present in OSS communities. But in the communities that have adopted such systems the actual influence and binding character is rather limited. In the PostgreSQL community, for example, though the voting system is regularly used, the outcome is merely "transferred to the 'to-do' list, so we know what has been decided on." Yet participants are still free to do what they want, irrespective of the outcome of the vote. "Not everyone agrees on [the vote]… people are still free to implement whatever they think is best."[16] The voting system of the Apache community is used for every new addition of source code. Yet, as in the PostgreSQL community, the vote is not used to reach a collective choice. Individuals are still free to remove the source code once it is voted in. When the outcome of the vote is that the source code should not be included, individuals are still free to create a new development line of which the source code is a part. In short, the voting systems hardly seem to result in a collective choice, that is, 'a choice that binds the collective.' Individuals in the community are still free to do something else.

Despite the relatively weak influence of the voting systems, some form of collective choice is needed. Participants in the communities have opportunity and even incentive to diverge and create variety (Van Wendel de Joode 2004a, Van Wendel de Joode et al 2003). Consider, for instance, the mechanism of parallel development lines, the openness of source code and the diversity among participants. In light of this potential to diverge, threats like inefficiency, incompatibility and fragmentation would seem to emerge (Egyedi & Van Wendel de Joode 2004, Kogut & Metiu 2001). Yet software developed in bigger OSS communities has an overall quality and market share that is relatively high, which hardly seems possible if the communities were unable to come to some degree of convergence and focusing of resources (e.g. Bauer & Pizka 2003, Kogut & Metiu 2001, Mockus et al 2002). The question is 'How, among the huge variety of software, are participants able to decide what software to adopt and what to ignore?' How is convergence achieved?

Part of the answer lies in the fact that individuals base their choices on *tags*. Tags create positive feedback. Individuals in the communities appear to mimic each other's behavior, which on a collective level gives rise to 'swarming' (for a discussion on swarming, see Kelly 1994). In swarms the participants select one patch of source code over another, or one open source license over many others. Many tags facilitate this process of mimicking and swarming. Important tags are, for instance: (i) the presence of statistics indicating the level of activity in a project. Such a statistics stimulate participants to select software that is also selected by many others. One respondent explained, "SourceForge has information on activity as well. If there are 20 different versions of a library for a certain purpose, and one has been downloaded 10,000 times and another one 20 times, it is clear which you choose first." Another mechanism is (ii) the level of reputation of the participants. Reputation may be "a way to attract attention from others" (Ljungberg 2000, p. 212). Other mechanisms are (iii) elegance of the source code, (iv) inclusion in software distributions like Red Hat and Debian and (v) listing on an Internet directory site like Freshmeat.[17]

---

[16] Both quotes are from an interview with the project leader of the PostgreSQL community.

[17] From the Internet: http://freshmeat.net/ (July 2004).

The empirical data in this research do not provide a clue as to which of these five tags is most dominant and important in attracting individuals. Conceptually, it is possible that each individual mechanism constitutes a sufficiently strong tag to attract many new participants and make certain communities popular. However, in line with an observation by West and O'Mahony (2005), with the rise of the number of OSS communities the competition for attention becomes greater. It is conceivable that it is increasingly difficult for new projects to attract new participants and users. This could mean that for new projects to become popular, they need to have a combination of tags.

Monitoring and graduated sanctioning

Monitoring and sanctioning in a community are needed to ensure that everyone adheres to the rules and codes of conduct that are agreed upon by the participants. In OSS communities monitoring and sanctioning ensure that participants adopt the coordination mechanisms, act according to the licenses and treat other participants respectfully. "Bad behavior is mainly rude behavior, swearing, consistently disrespecting other people's opinion."[18]

A first observation concerning monitoring and sanctioning is that many types of infractions are dealt with almost automatically. Due to a redundancy in mechanisms to deter conflicts, to support the development process and to prevent appropriation of open source software, many actions that could become a serious threat are actually automatically resolved. This redundancy in mechanisms reduces the potential threat of infractions and lowers some of the need for formal sanctioning mechanisms.

Participants in OSS communities have access to a limited number of more formal sanctioning mechanisms, but these are hardly used. In the Apache community participants could be "kicked out of the community." This means they lose their right to commit source code directly into the CVS or they are banned from a mailing list. However, this mechanism has never been used. "We could kick out people, but even that we have never done."[19] The fact that these sanctioning mechanisms are hardly used does not automatically mean that their role is limited. Their mere presence may be a sufficient deterrent for individuals to display grave counterproductive behavior.

The third observation is that the mere act of participating in OSS communities automatically leads to monitoring. In other words, participants incur few additional costs to monitor others, because (i) the processes and activities of participants in the communities are highly transparent (Osterloh 2002); (ii) participants are automatically notified of new activities, for example, through notification of changes to the CVS (Shaikh & Cornford 2003); and (iii) use of open source software automatically implies testing (Von Hippel & Von Krogh 2003).

The fourth observation is that sanctioning is the direct result of monitoring and participation. Programmers are sanctioned and stimulated to do things right in three different ways: (i) In the 'hall of blame' almost everything programmers write (e.g. e-mails and source code) is and remains visible to others. This ensures that participants have an incentive to do things right. (ii) Participants can create a fork when they strongly disagree – or are annoyed – with the general direction of a community, with its atmosphere or with individual decisions. The fork results in a destruction of value, as the same resources are now

[18] From an interview with a member of the steering committee of PostgreSQL.
[19] From an interview with a member of the ASF Board of Directors.

divided between two competing projects. The fork thus sanctions the participants in the community. (iii) Flaming, spamming and shunning (Maggioni 2002, Osterloh 2002) are other sanctions used to penalize participants. "If [their code is] inscrutable, sloppy, or hard to understand, then others will ignore it or pummel them with questions. That is a strong incentive to do it right" (Wayner 2000, p. 118).

Multiple layers of nested enterprise

The design principle multiple layers of nested enterprise draws attention to the diversity and interdependencies that exist in large and complex common pool resources. This is also true for OSS communities. Particularly the larger open source software programs consist of millions of lines of source code and can be used in combination with hundreds of different applications. Interdependencies must be managed to ensure its proper operation.

The first observation concerning this principle is that OSS communities have a highly developed division of labor (Koch & Schneider 2002). This division of labor is achieved in two ways, namely, through the modular design of the software, which results in the creation of many sub-communities, and through the division of activities.

Second, the division of labor and activities is emergent. Once again, there are no formal mechanisms by which tasks are identified and divided. No project leader can assign tasks to individuals. Instead, individuals choose for themselves what to work on. "[A]gents choose freely to focus on problems they think best fit their own interest and capabilities" (Bonaccorsi & Rossi 2003, p. 1247).

Third, the division of labor results in high degrees of task specialization. For instance, Mockus et al. (2002) demonstrated that of the top-15 bug reporters in the Apache community only three are also core programmers.

Fourth, the division of labor raises the level of efficiency in communities. The structure of OSS communities is layered, resembling the structure of an onion (Crowston et al 2003, Nakakoji et al 2002, Van Wendel de Joode et al 2003). The easier tasks, like reporting bugs or solving user problems, are carried out on the outside layers and the more difficult and challenging tasks are done closer to the center (Ye et al 2002). It is efficient for core developers to spend most of their time on the more difficult tasks and leave the easier ones to others. Indeed, they reportedly ignore simple tasks. "There are people who know practically everything. They won't react to trivial matters."[20] In contrast, less skilled programmers and even users perform the simpler tasks.

Fifth, the outside layers of the community constitute a learning environment, and as participants learn they tend to move inward (Edwards 2001, Ye et al 2002). This learning environment is based on the presence of many fairly simple activities, which allows users to get acquainted with the practices and processes in the community.

External recognition

The final design principle is external recognition. External recognition focuses on the interdependencies between a community and its outside environment. Essentially, the principle claims that no community can exist and survive without external authorities acknowledging and respecting the processes and structures in the community.

---

[20] From an interview with a Linux kernel developer.

Currently two debates surround open source software and the communities. Both debates are riddled with metaphors and stories. Furthermore, two extreme positions have emerged from them. The debates are relevant because they influence the perception that external authorities have of open source. The first debate is whether open source software is more secure or less secure than proprietarily developed software. The second debate is whether open source stifles or stimulates innovation in the software market. The root of both debates can be traced back to two underlying questions: (i) Is openness of the source code and the development process desirable? (ii) Are intellectual property rights on software desirable? Currently, the debates are undecided, making it as yet difficult for external authorities to act upon these issues.

In the meantime, participants in OSS communities have created and adopted mechanisms to counter some of the critique that their opponents have on the way in which software is created and maintained in the communities. The first mechanism is the Developer's Certificate of Origin. This certificate was adopted by the Linux community in response to the SCO case and the potentially disastrous consequences of sloppy IPR management. The certificate explicates the path the source code has gone through before its inclusion in the Linux kernel.[21] One goal of the certificate is to address industry concerns over intellectual property rights.[22] Appendix D presents a copy of the certificate. The second mechanism is the creation of the Linux Standards Base (LSB) and the Free Software Group (FSG). To become LSB Certified, software must undergo a formal documented testing procedure (Claybrook 2004), which will likely increase the credibility of that software. The third mechanism is external representation through foundations and project leaders. Many organizations want to reduce their uncertainty and prefer to have one clear point of contact, one entity with which they can do business. Yet the participants in open source communities change constantly and are located in many different locations. Project leaders and foundations have proven able to take up the contact person's role. The fourth mechanism is the use of disclaimers. The openness and perceived randomness of the software development process in the communities is due to the participation of both individual volunteers and organizations. One concern is that an organization will be sued if an employee has contributed sloppy source code that results in system crashes or security breaches. The disclaimer is intended to alleviate some of these concerns.

An interesting characteristic of the four mechanisms is that they do not conflict with the observations made regarding the other design principles. The mechanisms do suggest a level of institutionalization, which is different from the observations related to the other principles. However they do not interfere with the ability of individuals to act based on individual choices. In other words, the mechanisms do not interfere with or limit the freedom of individuals to make their own informed choices.

Conclusion

The first conclusion is that Ostrom's principles are a fruitful framework for analyzing OSS communities. Applying the principles has provided insight into the organizational processes in the communities.

---

[21] From an article on the Internet: http://news.com.com/Linux+contributors+face+new+rules/2100-7344_3-5218724.html (October 2004).

[22] From the Internet: http://www.nwc.com/showitem.jhtml?docid=1511buzz1 (October 2004).

Institutions can be identified in OSS communities. Consider, for instance, a voting system or project leadership. This research, however, arrived at the somewhat counterintuitive conclusion that many of the institutions in open source communities are 'light-handed.' Their influence on the actual behavior of individuals is surprisingly marginal. Some institutions, however, do have an important role in the communities. The role of the Apache Software Foundation in the Apache community, for instance, is not at all marginal. The ASF performs the important function of safeguarding participants in the community from future legal claims. Yet, by and large, many of the institutions in OSS communities perform only marginal roles in the communities' organization. One reason why is that, as reported by many respondents in this research, many participants ignore the institutions. They do what they think is best. They behave individualistically. Sometimes this means that they accept the choice of the project leader and the results of a voting system. In other situations, however, they might disagree and ignore these institutions. The previous pages identified and discussed a number of mechanisms that enable participants to ignore institutions and that allow them to act based on their own preferences.

Now we are confronted with two questions. First, what is the role of institutions? Possible explanations for the existence of institutions are: a) the institutions are an alternative explanation for the operation of the communities, b) the institutions are a way of acquiring external recognition, and, related to the above reasons, c) the institutions are a way of explaining the success of the communities.

Second, how are OSS communities organized if not through institutions? The functions described in the design principles are not solved through formal institutions. For instance, conflicts in OSS communities are not resolved through formal conflict resolution mechanisms, like voting systems or project leadership. Indeed, the potential negative effects of conflicts are resolved differently. This research indicates that the function of conflict resolution emerges out of individual choice and behavior. This brings us to an important conceptual point. Apparently, the design principles can be met through other means than formal institutional arrangements, namely through informal institutions or rather mechanisms that leave freedom for individual choice and behavior. These mechanisms combined with individual behavior aggregate into collective patterns of behavior – in such a way that the functions described in the design principles are solved.

But how does individual behavior aggregate into collective patterns of behavior? The simplest answer would be that every individual wants to collaborate and wants to act in the best interest of the community.

However, survey research proves otherwise. For one, research has demonstrated that many individuals participate in the communities primarily to receive personal benefits (e.g. Hars & Ou 2002, Hertel et al 2003). Furthermore, many choices and actions of participants could threaten the continuity of OSS communities. Participants, for instance, create new and competing alternatives even when they know their alternative will only be used by a very limited number of people; they send e-mails that contain no relevant information and merely consume other participants' time; and they frequently write source code that is inelegant, does not do what it is supposed to do and which creates many instabilities when added to the existing software.

The mechanisms identified in this research are part of an organization in which many potentially damaging actions and choices are made harmless. Stronger even, it seems as if many of these actions are turned into positive actions, into actions that profit a community as a whole. One good example is the design principle: 'conflict resolution mechanisms'. The fact that OSS communities have created mechanisms that allow participants to diverge and

create new and alternative solutions is believed to lead to innovation, competition and eventually better software.

This is not to say that divergence is always good. It implies for instance that processes in the communities are not necessarily efficient. There is overlap between developers, who frequently perform activities and tasks that have already been done by others. However, the mechanisms do ensure that conflicts do not threaten the continuity of the software development process and the community as a whole. And what is more, the communities have many mechanisms in place that ensure a certain degree of convergence to prevent an endless stream of divergence.

References

Arrow KJ. 1951. *Social Choice and individual values*. New York and London: John Wiley & Sons and Chapman & Hill

Axelrod R, Cohen MD. 1999. *Harnessing Complexity: Organizational Implications of a Scientific Frontier*. New York: Free Press

Bauer A, Pizka M. 2003. *The Contribution of Free Software to Software Evolution*. Presented at Sixth International Workshop on Principles of Software Evolution (IWPSE'03), Helsinki, Finland

Bekkers VJJM. 2000. *Voorbij de virtuele organisatie? Over de bestuurskundige betekenis van virtuele variëteit, contingentie en parallel organiseren (Past the Virtual Organization. The meaning for public administration of virtual variety, contingency and parallel organizing)*. 's Gravenhage: Elsevier bedrijfsinformatie

Benkler Y. 2002a. Coase's Penguin, or, Linux and the Nature of the Firm. *Yale Law Journal* 112: 369-446

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

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

Bollier D. 2002. *Silent Theft, the private plunder of our common wealth*. New York: Routledge

Bonabeau E, Dorigo M, Theraulaz G. 1999. *Swarm Intelligence: from natural to artificial systems*. New York: Oxford University Press

Bonaccorsi A, Rossi C. 2003. Why Open Source Software can succeed. *Research Policy* 32: 1243-58

Boyle J. 2003. The second enclosure movement and the construction of the public domain. *Law and contemporary problems* 66: 33-74

Bruns B. 2000. *Nanotechnology and the Commons: Implications of Open Source Abundance in Millennial Quasi-Commons*. Presented at "Constituting the Commons: Crafting Sustainable Commons in the New Millennium," the Eighth Biennial Conference of the International Association for the Study of Common Property, Bloomington, Indiana, USA

Claybrook B. 2004. Linux Standard Base: Enough Support? Why LSB compliance is important. In *LinuxWorld*, pp. 26-8

Crawford SES, Ostrom E. 1995. A grammar of institutions. *American Political Science Review* 89: 582-600

Crowston K, Scozzi B, Buonocore S. 2003. *An explorative study of open source software development structure*. Presented at Eleventh European Conference on Information Systems, Naples, Italy

Dafermos GN. 2001. Management and virtual decentralised Networks: The Linux Project. *First Monday. Peer reviewed journal on the Internet* 6: downloaded from the Internet: http://www.firstmonday.dk/issues/issue6_11/dafermos/ (December 2004)

Dalle J-M, Jullien N. 2003. 'Libre' software: turning fads into institutions? *Research policy* 32: 1-11

Deutsch M. 1973. *The Resolution of Conflict: constructive and destructive processes*. New Haven and London: Yale University Press

Dipboye RL, Smith CS, Howel WC. 1994. *Understanding Industrial and Organizational Psychology*. Fort Worth: Hartcourt Brace College Publishers

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

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

Fielding RT. 1999. Shared Leadership in the apache Project. *Communications of the Association for Computing* 42: 42/3

Franck E, Jungwirth C. 2003. Reconciling Rent-Seekers and Donators - The Governance Structure of Open Source. *Journal of Management and Governance* 7: 401-21

Gefu JO, Kolawole A. 2002. *Conflict in Common Property Resource Use: Experiences from an Irrigation Project*. Presented at "The Commons in an Age of Globalisation", the Ninth Biennial Conference of the International Association for the study of Common Property, Victoria Falls, Zimbabwe

Ghosh RA. 2005. Understanding Free Software Developers: Findings from the FLOSS Study. In *Making Sense of the Bazaar: Perspectives on Open Source and Free Software*, ed. J Feller, B Fitzgerald, S Hissam, K Lakhani. Cambridge: MIT Press

Hann I-H, Roberts J, Slaughter S, Fielding RT. 2002. *Why Do Developers Contribute to Open Source Projects? First Evidence of Ecomic Incentives*. Presented at Meeting Challenges and Surviving Success: 2nd Workshop on Open Source Software Engineering, 24th International Conference on Software Engineering, Orlando, USA

Hars A, Ou S. 2002. Working for Free? Motivations for Participating in Open-Source Projects. *International Journal of Electronic Commerce* 6: 25-39

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: 1159-77

Von Hippel E, Von Krogh G. 2003. Open Source Software and "Private-Collective" Innovation Model: Issues for Organization Science. *Organization Science* 14: 209-23

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

Holland JH. 1995. *Hidden Order: How Adaptation Builds Complexity*. Cambridge: Perseus Books

Jehn KA. 1995. A miltimethod examination of the benefits and detriments of intragroup conflict. *Administrative Science Quarterly* 40: 256-82

Jehn KA, Mannix EA. 2001. The dynamic nature of conflict: a longitudinal study of intragroup conflict and group performance. *Academy of Management Journal* 44: 238-51

Kelly K. 1994. *Out of Control; The new biology of machines, social systems and the economic world.* Cambridge: Perseus Books

Koch S, Schneider G. 2002. Effort, co-operation and co-ordination in an open source software project: GNOME. *Information systems Journal* 12: 27-42

Kogut B, Metiu A. 2001. Open-Source software development and distributed innovation. *Oxford Review of Economic Policy* 17: 248-64

Kollock P. 1996. *Design Principles for Online Communities*. Presented at Harvard Conference on the Internet and Society, Harvard, USA

Von Krogh G, Von Hippel E. 2003. Editorial: Special issue on open source software development. *Research Policy* 32: 1149-57

Kuwabara K. 2000. Linux: A Bazaar at the Edge of Chaos. *First Monday. Peer reviewed journal on the Internet* 5: downloaded from the Internet: www.firstmonday.dk/issues/issue5_3/kuwabara/ (December 2004)

Lakhani K, Von Hippel E. 2003. How Open Source Software Works: "Free" User-to-User Assistance. *Research Policy* 32: 922-43

Lanzara GF, Morner M. 2003. *The Knowledge Ecology of Open-Source Software Projects*. Presented at 19th EGOS Colloquium, Copenhagen

Lerner J, Tirole J. 2002a. *The Scope of Open Source Licensing*, NBER Working Papers 9363, downloaded from the Internet: http://ideas.repec.org/p/nbr/nberwo/9363.html (December 2004)

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

Ljungberg J. 2000. Open source movements as a model for organising. *European Journal of Information Systems* 9: 208-16

Maggioni MA. 2002. Open source software communities and industrial districts: a useful comparison? Unpublished research paper, downloaded from the Internet: opensource.mit.edu/papers/maggioni.pdf (November 2004)

Markus ML, Manville B, Agres CE. 2000. What Makes a Virtual Organization Work? *Sloan Management Review* 42: 13-26

McGowan D. 2001. Legal Implications of Open-Source Software. *University of Illinois Review* 241: 241-304

Mockus A, Fielding RT, Herbsleb JD. 2002. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology* 11: 309-46

Nakakoji K, Yamamoto Y, Nishinaka Y, Kishida K, Ye Y. 2002. *Evolution Patterns of Open-Source Software Systems and Communities*. Presented at International Workshop on Principles of Software Evolution (IWPSE 2002), Orlando, Florida

O'Mahony SC. 2003. Guarding the Commons: How Community Managed Software Projects Protect Their Work. *Research Policy* 32: 1179-98

Osterloh M. 2002. *Open Source Software Production The Magic Cauldron?* Presented at LINK Conference, Copenhagen

Ostrom E. 1990. *Governing the Commons; The Evolution of Institutions for Collective Action*. Cambridge: Cambridge University Press

Ostrom E. 1993. Design principles in long-enduring irrigation institutions. *Water Resources Research* 29: 1907-12

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

Raymond ES. 1999. *The Cathedral and the Bazaar: Musings on Linux and Open Source from an Accidental Revolutionary*. Sebastapol: O'Reilly

Resnick M. 1994. *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. Cambridge: The MIT Press

Rosenthal U. 1988. *Bureaupolitiek en Bureaupolitisme; om het behoud van een competitief overheidsbestel (inaugurele rede Leiden)*. Alphen aan de Rijn: Samsom H.D. Tjeenk Willink

Shaikh M, Cornford T. 2003. *Version control software for knowledge sharing, innovation and learning in OS*. Presented at Open Source Software Movements and Communities workshop, ICCT, Amsterdam

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

Torvalds L. 1999. The Linux Edge. *Communications of the Association for Computing* 42: 38, 9

Vemuri VK, Bertone V. 2004. Will the Open Source Movement Survive a Litigious Society? *Electronic Markets* 14: 114-23

Walker JM, Garzarelli G, Herr A, Ostrom E. 2000. Collective choice in the commons: experimental results on proposed allocation rules and votes. *The Economic Journal* 110: 212-34

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

Weber S. 2004. *The Success of Open Source*. Cambridge: Harvard University Press

Van Wendel de Joode R. 2004a. Conflicts in open source communities. *Electronic Markets* 14: 104-13

Van Wendel de Joode R. 2004b. *Continuity of the commons in open source communities*. Presented at 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 JA, Van Eeten MJG. 2003. *Protecting the Virtual Commons; Self-organizing open source communities and innovative intellectual property regimes*. The Hague: T.M.C. Asser Press

West J, O'Mahony SC. 2005. *Contrasting Community Building in Sponsored and Community Founded Open Source Projects*. Presented at 38th Annual Hawaii International Conference on System Sciences, Waikoloa, Hawaii

Ye Y, Kishida K, Nakakoji K, Yamamoto Y. 2002. *Creating and Maintaining Sustainable Open Source Software Communities*. Presented at International Symposium on Future Software Technology, Wuhan, China