Continuity of the commons in open source communities

Ruben van Wendel de Joode
PhD student
School of Organization & Management
Faculty of Technology, Policy and Management
Delft, University of Technology
PO Box 5015, 2600 GA Delft, The Netherlands
(T) 0031 15 278 1105, (F) 0031 15 278 6439, (E) rubenw@tbm.tudelft.nl
http://www.tbm.tudelft.nl/webstaf/rubenw/index.htm

Draft Paper

Paper submitted for the symposium of Santa Caterina on Challanges in the Internet and Interdisciplinary Research, Amalfi, Italy.

Abstract

This article focuses on the question: given that volunteers are motivated to participate in open source software development (e.g. Lakhani and von Hippel, 2000; Markus et al., 2000), what mechanisms are present to coordinate their invested time and effort and ensure continuity of software?
Elinor Ostrom (1990) identifies eight design principles, e.g. the presence of conflict resolution mechanisms and collective choice arrangements, that explain how individuals in communities ensure the continuity of depletable resources, like fishing grounds and grazing fields. This article provides a first discussion of the mechanisms that institutionalize these principles in open source communities.

Key words
Open source, software development, continuity and common pool resources

Continuity of the commons in open source communities

Introduction

This article discusses the methodology and some first results of a PhD research with the title: "The continuity in open source communities and open source software." This subject is intriguing and highly relevant for three reasons.

Open source software is developed and maintained by developers in Internet communities. The developers can be grouped in roughly two groups, namely: "professional hobbyists" and developers paid by commercial software and service providers (Markus et al., 2000; Weber, 2001). It is not difficult to understand how the difference in background will boost conflicts in open source communities: "One of the problems is that the individuals who develop on Apache who come from Sun have deadlines. That creates frustration, when they need to work together with people who do it for fun."[1] The rise of conflicts is increased by the fact that developers in the communities are mutually dependent: they simply cannot write complex software that consists of million lines of code by themselves. As developers have their own expertise, they are forced to cooperate and are dependent on the input of others (e.g. De Bruijn and Ten Heuvelhof, 2000, p. 12)

The second reason is the increasing importance of *continuity* of open source software, which is given by the fact that companies and governments are increasingly dependent on open source infrastructures and services. Consider for instance the New York Stock Exchange, which depends on open source software to support some of their mission-critical business processes.[2] Or consider IBM, a company that has invested billions in the development and marketing of open source software.[3] Furthermore, many governments worldwide stimulate the development and adoption of open source infrastructures and services.

The third reason is that characteristics in open source communities appear to run counter to the mechanisms prescribed in literature as to how continuity of goods, and software in particular, should be assured. The development and maintenance of software in open source communities is characterized by openness and an absence of central governance, which results in characteristics like openness of source code, lack of formal procedures, low entry barriers and lack of centralized ownership (e.g. McGowan, 2001; Sharma et al., 2002; Van Wendel de Joode et al., 2003). These characteristics are likely to conflict with mechanisms that enhance the likelihood of continuity, as described by theory.

These observations result in the question that is central in the research on which this article is based, namely: "*How is the continuity of open source software and open source communities achieved?*" This question can be answered from a number of perspectives. The perspective adopted in this article is based on institutional economics. One particular theory is used, namely theory from ongoing research on common pool resources (CPR's). There are two reasons for adopting this line of research:

- The fundamental problem in CPR research is the question how communities are able to assure their *continuity* and the *continuity* of the goods they provide and maintain (e.g. Ostrom, 1990; Ostrom, 1998; Ostrom, 1999; Ostrom et al., 1999; Ostrom, 2000)

[1] Cited in an interview with two members of the Apache Software Foundation.
[2] Based on an article on the Internet: http://www.it-director.com/article.php?articleid=2125 (November 2003)
[3] Based on an article on the Internet: http://news.com.com/2100-1001-275388.html?legacy=cnet (November 2003)

- An increasing amount of research has adopted the analogy with common pool resources to describe and analyze different issues regarding ICT networks and ICT products (e.g. Lukasik, 2000; Bollier, 2001; Hess and Ostrom, 2001; O'Mahony, 2003)

Structure of the article

This article will first introduce the theory of common pool resources. This part of the article will finish with an introduction of eight design principles, which are argued to explain how continuity is achieved in communities.
The open source communities are introduced and it will be explained that some of the issues facing source code development and maintenance in open source communities can be compared with the issues that face communities in which more "traditional" common pool resources are managed. Therefore, it is argued that research on common pool resources can provide a valuable framework to understand and evaluate the continuity in the development and maintenance of open source software.
The article will then continue with an exploration of one design principle, namely the design principle of conflict resolution mechanisms. This section sheds some light on how the research on common pool resources can be applied to open source communities. Furthermore, it provides a first demonstration of the differences and similarities that exist between the "traditional" common pool resource and the "new" common pool resources. It also provides a demonstration of the relevance of this and probably also other design principles.
The article will finish with a conclusion and a reflection.

Theoretical framework

The problem

Economics typically identifies four different types of goods, which can be characterized along two axes: excludability and jointness of consumption (Levacic, 1991). Common pool resources (CPR's) are typically goods (1) from which people cannot be excluded and (2) of which one person's consumption reduces the amount available to others (Ostrom, 1990; Levacic, 1991; Thomson and Schoonmaker Freudenberger, 1997; Kollock, 1999; Ostrom, 1999; Ostrom, 2000). Examples of CPR's are fishing grounds and clean air. These resources are typically faced with two problems:

1. The *collective action problem* stems from the fact that it is not possible to exclude people from consuming the good. Rational acting individuals have no incentive to produce and/or maintain the common pool resource, but each individual is better off when the good is produced and/or maintained (Olson, 1965; Ostrom, 1990).
2. The resources are also faced with a threat that is known as *Tragedy of the Commons* (Hardin, 1968). Every individual in a common pool resource will have an incentive to consume as much as possible. Consumption will limit the amount of resources available to others due to their subtractive nature. Especially natural resources, like fishing grounds, have a natural rate of regeneration. The resource, i.e. fishing ground, will be depleted if the rate of consumption exceeds the regeneration rate.

The origin of CPR research

Economics has long been focused on two dominant ways to overcome the two problems facing common pool resources, namely (1) to divide the good into smaller pieces, which are protected

by property rights and then let the market do the work; or (2) to impose hierarchical control (Ostrom, 1990). Elinor Ostrom (1990), however, argued differently. She analyzed many different communities from various parts of the world and realized that there is a third way for people to solve the collective action problem and to prevent a tragedy of the commons. She discovered that people who are organized in self-regulating communities succeeded, under certain conditions, to overcome the problem of collective action and ensure the continuous development and maintenance of common pool resources.
She wrote her findings in a book called *Governing the Commons; The Evolution of Institutions for Collective Action* (1990). 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 Analysis 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.

The design principles

CPR research suggests that sustainable commons management can be explained by the presence of 8 design principles (Ostrom, 1990; Kollock, 1996; Ostrom, 2000; Hess and Ostrom, 2001; Sekher, 2001).[4] Such a principle is

> an essential element or condition that helps to account for the success of these institutions in sustaining the CPRs and gaining the compliance of generation after generation of appropriators to the rules in use. (Ostrom, 1990, p. 90)

Thus, the design principles are essential conditions, which means that they must be present in each community that is able to achieve continuity of the resource. The absence of one of the principles is likely to result in a tragedy of the commons. The eight principles are listed in table 1.

Table 1 - Design principles of long-enduring self-organizing communities[5]

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

[4] Other line of research on common pool resources argues that the self-organizing commons should be explained along 7 rules (Ostrom et al., 1994). The difference between the design principles and the rules is that the rules apply to the individuals. The idea is that individuals act according to roles, roles that are necessary for the functioning of the community as a whole. The principles on the other hand address the issues, which the communities are faced with on the level of the community. This is independent of the roles of the individuals. First research on the communities indicates that the latter is more relevant to the free software communities, as the volunteers in the communities do not behave according to fixed roles.
[5] Adapted from Ostrom (1990)

The first design principle is: *clearly defined boundaries*. With clearly defined boundaries the members of the community can be sure that no "outsider" will reap the benefits of their efforts. Boundaries thus reduce the possibilities of free riding, which should enhance the motivation to cooperate. Boundaries are also important because they stimulate group members to interact more frequently (Kollock and Smith, 1996), which should stimulate cooperation and coordination as well.

The second design principle is *congruence between appropriation and provision rules and local conditions*. The community members should define (1) appropriation rules, which regulate the number of units of the resource that every insider is allowed to consume, and (2) provision rules, which regulate what every insider should contribute to the development and maintenance of the resource. These rules should be in congruence with the nature of the good and the local conditions. Generally, the more complex the community and good, the more difficult it will be to achieve this congruence.

The presence of *collective choice arrangements* is the third design principle. This rule addresses the need for mechanisms that enable decision-making in which every community member is involved and to which they should adhere. One of the issues that involve each group member, are the rules that govern the behavior of the community members. Generally group members will understand the problems facing the community and the resource best. Therefore they should be able to adjust the rules. This will increase the chance of a better fit between the rules and the local conditions.

The fourth and fifth principle are: *monitoring* and *violators of the rules are sanctioned gradually*.[6] Monitoring and sanctioning is needed to ensure compliance to the operational rules of the community. It is important that group members perform this monitoring themselves and that sanctions for first-time violators remain low. By keeping the first sanctions low, group members acknowledge the fact that in certain situations infraction might be acceptable or even necessary.

The presence of *conflict resolution mechanisms* is the sixth principle. Conflicts will always occur. Certain rules for instance will eventually be debated and such a debate should be possible. Without conflict resolution mechanisms these debates and conflicts can easily frustrate development and maintenance of the resource and might cause in a depletion of the resource and the end of the community (Smith, 1999).

The seventh principle is that *external authorities do not challenge the rules of the community*. To be sure that the rules are effective, the principles should be embedded within a larger, legal, context. Community members are less likely to obey the rules of the community if they know that external government authorities do not acknowledge the rules.

The eighth principle is *multiple layers of nested enterprises*. This principle addresses the need of the community to organize differently in different situations. This principle is especially important when the common pool resource is large and complex. If this is the case then this principle addresses the need for different rules and a different structure.

Open source communities

Open source communities connect many thousands of people who have different backgrounds and nationalities. Programmers, translators, testers and authors from almost every region in the world contribute to the development of open source software, sometimes without seeing each other in real life. Some of them spend four to then hours a day on the development of open source software, whereas others only contribute once a week or even less, but no matter how much time and effort they put into developing open source code, there are still many developers who do not

[6] Ostrom (1990) also discusses principle 4 and 5 together.

get paid for their work. Most developers are volunteers that are intrinsically motivated to spend their time on developing open source code (Markus et al., 2000).
The next section will discuss the resource in open source communities and will discuss why this resource can be compared to the more "traditional" common pool resources, like fishing ground and grazing grounds.

The role of source code

In open source communities the most important product that is created is without a doubt the software. The developers focus on managing the source code of the software, which is the human-readable part of it. The fact that the source code is open allows developers to understand the working of software and it allows them to make modifications to the software. The source code is needed to improve, enhance and write new software.
Source code is created by people and is a product of the human mind. As such, it is often referred to as intellectual property. Intellectual property is the collection of products from the human mind that are regulated by law. Examples of intellectual property are literary and artistic works (including computer programs), technical inventions or trademarks and trade names (Van Wendel de Joode et al., 2003).

Source code as a common pool resource

Others have applied the ideas from research on common pool resources to information goods on the Internet (e.g. Lukasik, 2000; Bollier, 2001; O'Mahony, 2003). Hess and Ostrom (2001) provide a critical argument on the applicability of CPR to information goods. They do belief that the way of thinking as described in CPR research is relevant. To assess whether this is true we should answer the question: Can open source code be compared to the resources that are analyzed in CPR research? To answer this question we need to look at two aspects, namely (1) whether people can be excluded from the consumption of open source code and (2) whether the consumption of open source code is subtractive.
The first issue of exclusiveness is rather straightforward, as it is impossible to exclude people from using the source code in open source communities. The source code is licensed by open source licenses that basically allow anyone to use and modify the source code (Perens, 1999; Markus et al., 2000; Kogut and Metiu, 2001; McGowan, 2001; Lerner and Tirole, 2002a; Stallman, 2002). Therefore one could say that the source code in free software communities is non-excludable.[7]
The question whether the consumption of source code is subtractive, is more difficult to answer. On first sight one would be inclined to say that the consumption is more joint than subtractive, as anyone can download and use the source code without reducing the amount of source code available to others. But, one can also argue why consumption can be subtractive and why source code can also be regarded to be a common pool resource.
Let's consider the common pool resources analyzed in most CPR research, which primarily focuses on natural resources. One example is a fishing ground. Taking fish from the resource diminishes the amount of fish left for others; hence consumption is subtractive. The fish is taken from the resource and becomes the private property of the fisherman who took it. He now owns the fish and can exclude others from its consumption or can decide to sell the fish to anyone who

[7] Another reason why exclusion is difficult is the fact that people cannot check whether or not others have used some of the intellectual property of the communities. Many companies and individuals do not reveal their source code. For the communities it would be possible to exclude people from using and consuming the source code by not revealing it, but this is not done in the communities.

is interested in buying. Too many people appropriating too much fish from the fishing ground is likely to lead to a tragedy of the commons.
A comparable tale can be told for intellectual property in the communities. Let us consider the example of AT&T and Unix.[8] Initially AT&T made the source code of Unix available to anyone that had an interest in it. People from different parts of the world could view the code and could make amendments to it. So a community started to form in which the source code was managed as if it were under a common property regime. AT&T then took the source code out of the commons by claiming legal ownership on part of the source code. And indeed, courts judged that AT&T owned a large part of the source code and that they were allowed to exclude others from its use. In that sense their appropriation of the source code diminished the source code available to others. Thus, viewed in this way the consumption of source code, when combined with an intellectual property claim, is subtractive and could cause a depletion of the intellectual property available in the commons.
In open source communities the event that a company claims private ownership on a piece of source code is referred to as hijacking:

> If a commercial vendor hijacked a community managed project, the future stream of benefits that would stem from the collective resource would be made unavailable to the community. This type of problem shares some features with nonrenewable resources or common pool resource problems. (O'Mahony, 2003, p. 5)

To summarize open source code shares characteristics of a common pool resource, because people cannot be excluded from its consumption and the future streams of benefits are at risk (O'Mahony, 2003).

Research approach

Open source code shares some of the characteristics of the more "traditional" common pool resources. Thus, according to CPR research, the eight principles can explain how open source communities ensure continuity of the community and the resource.[9] Therefore, each principle is analyzed and for each principle the following questions will be answered:

1. Does the principle address an issue that is relevant to open source communities?

2. Is the principle present in open source communities;
    a. If so, how is the principle implemented?
    b. If not, what problems can we expect and are these problems solved in a different way?

To answer these questions several techniques are used: (1) analysis of documents, websites and mailing lists; (2) study of secondary literature; and, in particular, (3) face-to-face (semi-structured) interviews with experts actively involved in open source development.

[8] The example is greatly simplified and is primarily based on the account that is documented in Wayner (2000). Currently, another example is frequently cited in the media, namely the case of SCO against IBM. SCO claims that part of the Linux kernel is their property and demands that every user of Linux starts paying a licensing fee. It is not difficult to understand how this case could jeopardize the continuity of Linux and other open source projects.
[9] The eighth design principle *multiple layers of nested enterprise* is less important and can be absent when the common pool resource is very small. However, if the resource is big and complex then this principle is likely to be found. (Ostrom, 1990)

The interviews took place from October 2001 to August 2002. Forty-eight respondents from the Netherlands, the US and Germany were interviewed. They can roughly be grouped into two categories. The category of *engineers* consists of people who are actively involved in open source development. Most of them develop and revise open source software. Some perform other activities, like writing manuals, translating or testing software. The other category is that of the *managers*. It largely consists of company managers who have either implemented open source to support business processes or sell open source commercially. Table 2 presents the distribution of respondents according to country and category.

Table 2 –Respondents according to country and category

	Engineers	Managers	Total
Germany	2	3	5
Netherlands	15	5	20
US	22	1	23
Total	39	9	48

The sixth design principle: conflict resolution mechanisms

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 way in which continuity is achieved in the communities, this article will focus on just one of the eight design principles, namely *conflict resolution mechanisms*.

First, the characteristics of conflicts in open source communities will be identified. It will be demonstrated how conflicts can result endanger the continuity of software development and maintenance, which seriously threatens the continuity of open source communities and the source code. The second section will discuss one often-cited conflict resolution mechanism, namely third-party intervention. This section will argue that this is not the dominant way of conflict resolution in the communities.

The sections three, four and five will introduce three mechanisms, which minimize the negative effects of conflicts in open source communities. Due to these mechanisms, conflicts in open source communities are no longer threatening the continuity of open source communities and the source code, but the conflicts are transformed into a valuable source of creativity and vitality. These mechanisms are (1) modularity, (2) parallel development lines and (3) the exit option.

The characteristics of conflicts in OS communities

Most conflicts in OS communities have a win-lose character, which is "the situation in which more for one means less for the other." (Schelling, 1956, p. 281) This becomes especially visible in conflicts between programmers that arise due to a disagreement about the quality and appropriateness of a certain piece of source code. Consider a situation in which one programmer or a group of programmers argues how a certain piece of source is good and should thus be included. Other programmers, however, might - for different reasons - argue how the source code is not very good or is inappropriate and should not be included. Initially, the conflict has a win-lose character: the source code is included or it is not. There is no middle ground or win-win situation.

Another example of a win-lose conflict would be the situation in which someone implements a change to the source code. Other developers disagree with that change. Instead, of starting an argument, the other develops creates an alternative and replaces the original source code with the alternative. The creator of the original change is likely to disagree with the change and might

decide to override the latest change with the initial change, which is the start of a so-called "commit-war" (McCormick, 2003). In a commit-war the programmers do not use arguments and words to convince other programmers, but simply act according to their own ideas and interests. Commit-wars are an expected part of a project where "you have 300 people changing files all over, millions of files." (McCormick, 2003, p. 19)

The described examples have a win-lose character and are likely to have at least the following negative effects:

- *Overt hostility*. One negative effect of a win-lose conflict could be that it leads to "strong negative feelings, blindness to interdependencies, and uncontrolled escalation of aggressive action and counteraction." (Brown, 1984, p. 378)
- *Inactivity*. Another negative effect could be that it results in a situation in which everybody is actively pursuing their own interests and protecting their own values, which will obstruct the decision-making process. This situation can best be understood as a trench war (Rosenthal, 1988).

In open source communities conflict resolution mechanisms must be present to mitigate or solve win-lose conflicts to overcome the negative effects of these conflicts.

Third-party intervention in OS communities

One frequently mentioned mechanism to resolve conflicts is by third-party intervention (e.g. Elangovan, 1998; Nugent, 2002). Two types of third-party intervention are relevant here, namely mediation and arbitration.[10] *Mediation* in conflicts means that an organization or individual intervenes between the conflicting parties. This third party is called a mediator. He "is an impartial outsider who tries to aid the negotiators in their quest to find a compromise agreement." (Raiffa, 1992, p. 23) The role of an *arbitrator* is somewhat comparable with that of a mediator, but differs on one important aspect. Whereas the mediator has no authority to impose a solution the arbitrator does have such authority (Raiffa, 1992). The next section will investigate whether or not mediation and arbitration are actually present in open source communities. To do so this section will focus on the Linux community.

Linus Torvalds was the founder of Linux and still maintains what is considered to be the standard version of the Linux kernel. Much research addresses the puzzling role of Linus Torvalds in the Linux community. Some argue that he is a clear leader (e.g. Moon and Sproull, 2000; Lerner and Tirole, 2002b). Others feel that his role is more moderate (e.g. Van Wendel de Joode et al., 2003; von Hippel and Von Krogh, 2003). Whatever, his role may be, one thing that appears to lack in current research is the possibility that he is a mediator or an arbitrator.
Let us consider an example of a conflict, which centered on a specific part of the Linux kernel, namely Virtual Memory. There are two alternatives: one is maintained by Rik van Riel and the other by Andrea Arcangli. The conflict manifested itself for the first time in May 2000. Since then, the conflicting parties have had many clashes, which can be observed on the Linux kernel

[10] Raiffa (Raiffa, 1992) adds two other forms of third-party intervention, namely intervention by a facilitator and intervention by a rules manipulator. These two forms of intervention are not further analyzed in this section. The reasons for choosing to do so are relatively straightforward. First, the facilitator is used to get the relevant parties to talk to each other. The problem, however, in OS communities is not that the involved parties do not talk to each other. On the contrary most conflicts are heavily discussed and debated on the mailing lists. Intervention by a rules manipulator is also not further analyzed, but is considered to be a specific form of arbitration.

mailing list in the period from May 2000 to December 2001.[11] Occasionally Linus Torvalds has interfered in these discussions. When he did, he became an active participant displaying his preferences just like any other participant. Take for example the following statement by Linus Torvalds: "I still don't like some of the VM changes, but integrating Andrea's VM changes results in (a) better performance and (b) much cleaner inactive page handling in particular."[12]

Based on this and other statements, it is difficult to support the claim that Linus Torvalds is a *mediator*. He clearly displays his own opinion and becomes one of the many programmers who are involved in the conflict. Although this is just one example, there are many more of such examples in which Linus Torvalds is not afraid to make his opinion heard. Therefore, he is not a mediator.
It is argued here that Linus is neither an *arbitrator*. Returning to the example of Virtual Memory, the fact that Andrea's virtual memory has driven Rik's version out of the official version of the Linux kernel, could be reason to believe that he is an arbitrator, because Andrea's virtual memory has long been the preferred choice of Linus Torvalds. There are, however, many reasons that explain why Linus is not an arbitrator, either. The most obvious reason is the fact that if Linus really were an arbitrator, the conflict between Andrea and Rik would never have continued for so long. Linus made his opinion heard on many occasions and in many instances he expressed his preference for Andrea's system, but that never resolved the conflict. The conflict continued with or without the interference of Linus Torvalds and therefore it is safe to conclude that his role in resolving the conflict was rather limited.
According to the president of Linux International, Linus Torvalds should not be considered a mediator or arbitrator of conflict. He actually argues how Linus is known to stimulate the rise of conflicts rather then try and solve them.[13]

To summarize conflicts in the Linux community and other open source communities are not resolved due to the interference of a third party.[14] Mediation and arbitration play a minor role in the resolution of conflicts, at the very best. Two observations lead to the conclusion that mediation and arbitration are not the primary source of conflict resolution, they are:

- Many conflicts are solved without the interference of a third party. Apparently, neither mediation nor arbitration resolves these conflicts. These conflicts are thus solved in another way.
- There is also no significant evidence that supports the claim that conflicts were resolved through mediation or arbitration.

The role of modularity in conflicts

Most open source software is extremely *modular* (e.g. Kogut and Metiu, 2001; McKelvey, 2001; Benkler, 2002; Bonaccorsi and Rossi, 2003). Consider the following statement from an active Linux developer:

> 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

[11] Many examples of these clashes can be found on the Internet. An example can be found here: http://kt.zork.net/kernel-traffic/kt20010112_102.html#10 (last visited April 14th, 2003)
[12] Taken from the Internet: http://kt.zork.net/kernel-traffic/kt20011001_135.html#4 (last visited June 27th, 2003)
[13] Based on an interview with the president of Linux International.
[14] It is beyond the scope of this article to provide examples of other open source communities.

> into ten pieces and if you manage to provide them with good interfaces then you only need to understand the separate pieces.[15]

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 and Tirole, 2002b, p. 28). Modularity results in fewer conflicts for two reasons.

It increases *independence* between programmers and hence reduces the amount of conflicts. Developers 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."[16] Therefore, developers 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 developers 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, developers 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.

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 stimulates creativity. It also provides a countervailing force against people that want to enforce their decision on others and 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."[17]

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 and 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.

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. Nowhere is the diversity between volunteers in the

[15] Translated from Dutch.

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

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

communities as clear and obvious as the diversity between non-technical contributors and highly skilled and trained software developers. 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.[18] On the other hand the highly skilled and trained professionals 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 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 developers, 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.

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.

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

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

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 exit option is institutionalized in many different ways in open source communities. 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."[19]
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 and 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 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!"[20]

Conclusion and reflection

This article has introduced research on common pool resources. Furthermore, it has argued why open source code can be compared to a common pool resource and as such can be analyzed through the same framework. Applying the framework of CPR on open source communities can contribute to:

A An increased understanding for the factors that influence the continuity of open source software development and maintenance.

Applying the framework of CPR research can provide an insight in the issues that surround open source software development in general and the continuity of open source code in particular. Decision-makers from companies can use this insight to better understand the threats that face the continuity of open source and to better understand how they can contribute to the continuity of open source software development and maintenance. Furthermore, it can make decision-makers and open source developers understand that certain alleged negative aspects can have a positive influence on the continuity of the source code.
This article provided the example of forking. Research suggests that forking is an event that demonstrates the flaws in open source software development and that should be prevented at all costs and at all times. Using the perspective of CPR research sheds a different light on this issue

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

and enables us to understand that forking is a very efficient and effective means to prevent conflicts from endangering the continuity of open source communities. Using the perspective of CPR research can make open source developers and decision-makers understand that they should not try and ban the act of forking all together, but much rather should find ways to assess each situation individually and decide whether a fork is really necessary.

B An increased understanding of the way in which CPR research can be applied to ICT networks and ICT goods.

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 some nice analogies. Hess and Ostrom (2001) try to take the discussion one step further. In their paper, which appears to be primarily based on a theoretical investigation, 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 believe. 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.

Reference List

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

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

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

Brown, L. D. (1984) Managing conflict among groups, in *Organizational Psychology, a book of readings*, Kolb, D. A., I. M. Rubin and J. M. McIntyre (Eds) Prentice-Hall, Englewood Cliffs.

De Bruijn, H. and E. Ten Heuvelhof (2000) *Networks and Decision Making*, LEMMA Publishers, Utrecht.

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

Elangovan, A. R. (1998) Managerial intervention in organizational disputes: Testing a prescriptive model of strategy selection, *International Journal of Conflict Management,* 9 (4) 301-335.

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

Hess, C. and E. Ostrom (2001) Artifacts, Facilities, And Content: Information as a Common Pool Resource, in *Conference on the Public Domain*, Duke Law School, pp. 44 - 79.

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

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

Kollock, P. (1996) Design Principles for Online Communities, in *Harvard Conference on the Internet and Society*, Harvard.

Kollock, P. (1999) The economies of online cooperation: Gifts and public goods in cyberspace, in *Communities in Cyberspace*, Smith, M. A. and P. Kollock (Eds) Routledge, London, pp. 220-239.

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

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

Lakhani, K. and E. von Hippel (2000) How Open Source Software Works: "Free" User-to-User Assistance, in *MIT Sloan School of Management Working Paper #4117*, Boston.

Lerner, J. and J. Tirole (2002a) The Scope of Open Source Licensing.

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

Levacic, R. (1991) Markets and government: an overview, in *Markets, hierarchies & networks; the coordination of social life*, Thompson, G., J. Frances, R. Levacic and J. Mitchell (Eds) The Open University, London, pp. 35 - 65.

Lukasik, S. J. (2000) Protecting the global information commons, *Telecommunications Policy,* 24 519-531.

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

McCormick, C. (2003) "The Big Project That Never Ends": Role and Task Negotiation Within an Emerging Occupational Community, in *Department of Sociology*, University of Albany,, pp. 43.

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*, Centre for Research on Innovation and Competition, The University of Manchester, Manchester.

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

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

Nugent, P. S. (2002) Managing conflict: Third-party interventions for managers, *Academy of Management Executive,* 16 (1) 139-154.

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

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*, The political economy of institutions and decisions, Cambridge University Press, Cambridge.

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.

Ostrom, E., J. Burger, C. B. Field, R. B. Norgaard and D. Policansky (1999) Revisiting the Commons: Local Lessons, Global Challenges, *Science,* 284 278-282.

Ostrom, E., R. Gardner and J. Walker (1994) *Rules, Games, and Common-Pool Resources*, The University of Michigan Press, Ann Arbor.

Perens, B. (1999) The Open Source Definition, in *OpenSources Voices from the Open Source Revolution*, DiBona, C., S. Ockman and M. Stone (Eds) O'Reilly & Associates, Sebastopol, pp. 171-189.

Raiffa, H. (1992) *The art and science of negotiation*, The Belknap Press of Harvard University Press, Cambridge.

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

Schelling, T. C. (1956) An essay on bargaining, *The American Economic Review,* 46 (3) 281-306.

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

Sharma, S., V. Sugumaran and B. Rajagopalan (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 *Communities in Cyberspace*, Smith, M. A. and P. Kollock (Eds) Routledge, London, pp. 134-163.

Stallman, R. (2002) *Free Software, Free Society: Selected Essays of Richard M. Stallman*, GNU Press, Boston.

Thomson, J. T. and K. Schoonmaker Freudenberger (1997) *Crafting institutional arrangements for community forestry*, Food and agriculture organization of the United Nations, Rome.

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

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

Weber, S. (2001) The Political Economy of Open Source Software, *Open Source Working Paper for the Global Business Network*.