Managing Conflicts in Open Source Communities

RUBEN VAN WENDEL DE JOODE

INTRODUCTION

More and more companies have become dependent on open source (OS) software. One of the best examples is probably the New York Stock Exchange (NYSE). In 2001 the NYSE decided to adopt OS software to support some of its mission critical business processes.1 Other examples of companies that have adopted OS software are Shell, the French daily Le Figaro and the US Army.2 All of these companies are truly not small and yet they are willing to support their processes on software that is primarily developed by individual programmers who are collaborating in communities on the Internet.

The fact that an increasing number of companies have adopted OS software has not only resulted in an increase in (media) attention for OS communities and their development model. It has also led to an increase in the number of companies that consider OS software as a potential source of revenue. Next to many small and new companies that have entered the software market, companies like IBM and SUN have declared their interest and their involvement in a number of OS software projects.

The fact that more and more companies have a commercial interest in and base their business model on OS software has an effect on the structure and processes of OS communities (e.g. Van Wendel de Joode et al. 2003). Companies will for instance hire programmers and pay them ‘so that they can work on OSS projects full-time’ (Hertel et al. 2003, p. 1160). These programmers are expected to meet deadlines and minimal levels of quality, etc. In other words, they have to deliver qualitatively high work and be time- and cost-efficient at the same time. Time- and cost-efficiency especially are interests that are different from the interests of voluntary programmers, who still dominate the development of open source software (Hertel et al. 2003; Hars and Ou 2002). Voluntary programmers participate because they want to earn a reputation (Lakhani and Von Hippel 2000; Raymond 2001), want to learn (Von Hippel and Von Krogh 2003), to solve a personal need (Wayner 2000; Hertel et al. 2003) or to have fun (Markus et al. 2000; Torvalds and Diamond 2001; Hertel et al. 2003).

Brian Behlendorf, who has been involved in Apache since the start in 1995 and who is still one of the board members of the Apache Software Foundation, also recognized this difference in interest and states:

You could say there is a difference between a developer from for example IBM and a private developer who works at it for his hobby. The first will be working more structural towards deadlines and
Much literature has been written that claims that differences in interest will inevitably result in conflicts (Amason 1996; Brown 1984; Deutsch 1973; Jehn 1995; Pruitt 1998). And indeed programmers do report that conflicts arise due to the involvement of paid programmers: ‘One of the problems is that the individuals who develop on Apache and come from Sun have deadlines. That creates frustration, when they need to work together with people who do it for fun.’ Conflicts can have numerous advantages. High levels of conflicts or specific types of conflicts can, however, threaten the speed of decision-making, hinder implementation (Jehn and Mannix 2001) and even threaten the continuity of communities (Ostrom 1990, 1998).

This paper will identify and discuss mechanisms that have a role in the management of conflicts in open source communities. The goal is to identify mechanisms that companies can potentially deploy to ensure the realization of their commercial interests. The problem question reads as follows: What mechanisms are relevant for the management of conflicts in open source communities, that can also be used by companies to secure their own, commercial, interests?

Relevance

There are a number of reasons why conflict management in open source communities is a highly relevant and interesting topic. First, previous research suggests that success of teams highly depends on the way they manage conflicts: ‘Consistent with previous research on traditional teams, we expect that a major factor in the success or failure of distributed teams’ performance will hinge on the way they manage conflict’ (Mannix et al. 2002, p. 1252; see also Montoya-Weiss et al. 2001). This implies that programmers in successful open source communities, like Apache, must have been able to manage their conflicts. Current research on open source communities has hardly addressed this issue yet.

Second, more and more companies are becoming involved in the development of open source software. To secure their commercial interests they need to identify and deploy arrangements enabling them to manage the conflicts between their programmers and the voluntary programmers in the communities. It is believed that this is a major puzzle for companies to solve, that is, if they want to secure the achievement of their commercial interests.

Structure of the paper

The first part of this paper provides a theoretical discussion of conflicts. Next, the methodology underlying the paper is presented. The third part is an analysis of conflicts in OS communities. It is argued that both task conflicts as well as affective conflicts are present. Then, four mechanisms are presented and discussed. Empirical examples, which provide some first evidence of the potential relevance and presence of each of the mechanisms, are presented. The fifth part of the paper is an example of a company that has institutionalized parallel development lines to secure its commercial interests. The paper finishes with some concluding remarks.

CONFLICT

Conflicts defined

According to Pruitt (1998) the definitions of conflict found in literature can be divided into two categories. The first is ‘conflict behaviour — opposing action taken by two or more parties’ (p. 470) and the second refers to ‘one or another source of conflict behaviour, most commonly divergence of interest and annoyance that is attributed to another party …’ (p. 470). The definition used in this paper is borrowed from Jehn (1995, p. 257), who writes: ‘Conflict has been broadly defined as perceived incompatibilities (Boulding, 1963) or perceptions by the parties involved that they hold discrepant views or have interpersonal incompatibilities.’ This definition of conflicts is focused on the source of conflict behaviour, which falls in the second category of definitions, as identified by Pruitt.

Literature on conflicts generally makes a distinction between two types of conflicts. The first is task conflict, which is ‘an awareness of differences in viewpoints and opinions pertaining to a group task.’ (Jehn and Mannix 2001, p. 238) This type of conflict is considered to be functional (e.g. Amason 1996), because they arise when people have different opinions about how a specific task should be performed. Such a difference in opinion will increase the general understanding of the task at hand and will positively contribute to task performance (Jehn 1995).

The second type of conflict is affective conflict. This type of conflict is emotional and focuses on interpersonal incompatibilities. It includes ‘affective components such as feeling tension or friction’ (Jehn and Mannix 2001, p. 238) Research has demonstrated that this type of conflict is dysfunctional and is a threat to group performance (Amason 1996; Jehn 1995).

Advantages and disadvantages of conflicts

Back in 1976 Thomas already argued that conflicts can have positive effects: ‘More and more, social scientists are coming to realize — and to demonstrate — that conflict itself is no evil, but rather a phenomenon which can have
Conflicts can lead to disadvantages: Research has demonstrated that especially affective conflicts are primarily associated with task conflicts (e.g. Jehn 1995; Amason 1996). Research shows that conflicts can have the following advantages:

- **Higher level of productivity.** According to Likert and Likert (1976) research has proven that groups that regularly have conflicts achieve a higher level of productivity than groups that do not have conflict.
- **Prevents groupthink.** Task conflicts stimulate group members to thoughtfully consider criticism and alternatives (Jehn 1995).
- **Increased creativity.** Task conflicts stimulate interest and curiosity and are therefore likely to lead to more creativity (e.g. Deutsch 1973).
- **Increased vitality.** According to Rosenthal (1988) to stimulate conflicts and competition in an organization will create checks and balances and will introduce countervailing forces. In this view, to have conflicts means that no party can easily enforce their interests, opinions and values on others, which will increase the stability of the collective. In other words the presence of conflicts — and this seems paradoxically — will lead to more stability and balance in the collective (see also Pondy 1967).

Next to advantages, conflicts can have disadvantages. Research has demonstrated that especially affective conflicts can be dysfunctional and can have the following disadvantages:

- Conflicts can become destructive and produce ‘strong negative feelings, blindness to interdependencies, and uncontrolled escalation of aggressive action and counteraction’ (Brown 1984, p. 378). Conflicts can lead to overt hostilities, which is obviously not beneficial to organizations.
- Conflicts can lead to inactivity. Everybody is actively pursuing their own interests and protecting their own values, which will obstruct the decision-making process and the implementation process (Jehn and Mannix 2001). This situation can best be understood as a trench war (Rosenthal 1988) in which no one is able to progress and the collective as a whole is at a stalemate.
- Conflicts can threaten the continuity of communities. Research on collective action has demonstrated that successful teams can be characterized by the ease with which one can access mechanisms to resolve conflicts. The continuity of teams is threatened when means to resolve conflicts are absent (Ostrom 1990, 1998).

### The management of conflicts

Much research argues that the performance of a team depends highly on the way they are able to manage their conflicts (e.g. Lovelace et al. 2001). Montaya-Weiss et al. (2001, p. 1252) argue: ‘we focused on conflict management because it is a fundamental issue for effective virtual teams’ performance, given the inherent communication and coordination challenges they face.’ The management of conflicts is, for a number of reasons, not a simple activity.

The first reason is that task conflicts and affective conflicts are correlated (Amason 1996). Under certain conditions, group members are reported to perceive comments related to tasks as personal criticism, which gives rise to affective conflicts (Mannix et al. 2002). If members continue to perceive task conflicts as affective conflicts then ‘the result may be a steady rise in both task and relationship conflict and a performance loss rather than gain’ (Jehn and Mannix 2001). The second reason is that there is an optimal level of task conflict, below or beyond which they are beneficial to group performance (Pondy 1967). Similarly, Brown (1984) states that too many task conflicts can result in bad group performance. The third reason is that task conflicts are not always beneficial to group performance. Jehn (1995) argues that groups performing routine tasks will not benefit from task conflicts; on the contrary, the presence of task conflicts is likely to decrease the performance of such groups.

Thus, conflict management is needed (a) to prevent task conflicts from giving rise to affective conflicts and (b) to prevent the level of task conflict from decreasing below or rising beyond the optimal level of task conflict, which is needed to successfully complete a certain task.

### METHODOLOGY

**Sources of data retrieval**

This paper presents the first results from an explorative study on open source communities. The data presented are based on an analysis of documents, websites and mailing lists, a study of secondary literature, and, in particular, on face-to-face (semi-structured) interviews with experts who are actively involved in open source software development. The interviews took place from October 2001 to August 2002. In that period 48 respondents from the Netherlands, the US and Germany were interviewed. Most of the statements cited in the remainder of this paper were cited during the face-to-face interviews that were held with the respondents.

The respondents can roughly be grouped into two categories. The category of engineers consists of people, primarily programmers, who are actively involved in OS 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. This category...
largely consists of company managers who have either implemented OS to support business processes or sell OS commercially. Table 1 presents the distribution of respondents according to country and category.

The respondents that were interviewed were active members in a wide range of different communities. The communities most frequently mentioned were Linux, Apache and Debian.

Limitations of this research

One limitation of this research is that it focuses on larger and more successful open source communities. The two communities that receive most attention are Apache and Linux, which are not representative for the wide variety of small and unknown open source communities that are for instance listed on SourceForge. Therefore, the results of this explorative study can and should not be generalized to other open source communities.

Furthermore, the empirical data presented in this paper are based on a small number of interviews and on a limited amount of discussions and statements found on mailing lists. The results are not based on rigorous empirical research and therefore it is not suitable for reaching generalizable conclusions about conflicts in open source communities. The paper presents the results from a highly explorative research on conflict management in open source communities.

CONFLICTS IN OPEN SOURCE COMMUNITIES

It is argued here that both task conflicts and affective conflicts can be identified in open source communities. The presence of affective conflicts suggests that some form of conflict management is needed to prevent it from endangering the continuity of the communities.

Task conflicts in open source communities

Task conflicts are present when people are aware that they experience differences in viewpoints and opinions regarding a group task (Jehn and Mannix 2001). This section presents a number of statements, which were made during the interviews conducted with programmers from open source communities. They clearly demonstrate their awareness of these kinds of differences.

An active programmer in the Apache community who is paid to maintain the software for his company explains: ‘we have a huge experience running major websites and we give back this experience to the community … we are a user and we have therefore a different perspective. We are much more pragmatic and we contribute in giving cases or certain situations where Apache can be improved, based on real business situations.’

A Linux programmer states the following: ‘frequently they are students between 19 and 29, who own a small computer on which they work. Some of them are extremely intelligent, but their programming activities are aimed at their own situation, namely small computers. Companies, however, have big systems with big databases and students have no experience with the problems these big companies experience.’

Finally, the project leader of the Python programming language argues: ‘most companies don’t want the language to change … Personally I would lose my interest if I could not continue to improve the language.’

What these three statements have in common is that they demonstrate awareness among programmers about their differences in viewpoints and backgrounds and hence they provide first evidence that task conflicts are actually present.

Affective conflicts in open source communities

There are also many examples mentioned by respondents that provide evidence that affective conflicts are present. A programmer from the OpenOffice community stated: ‘People who have no cultural or language skills cause problems. Native speakers would understand things that non-native didn’t understand and they would get pissed of. They became counter-productive … part of this was due to the US — French interface.’ This is an example of an affective conflict that is caused by interpersonal incompatibilities between programmers.

Two programmers from the Debian community gave another example of an affective conflict. They addressed the presence of former project leader Bruce Perens. During the interviews they argued that his autocratic leadership style used to cause many conflicts in the Debian community.

A third example can be found in a statement by Zack Brown who provides weekly summaries of the Linux kernel mailing lists: ‘Rik van Riel and Andrea Arcangeli just don’t get along … Andrea said, “Your arguments are senseless.” Rik finally took the bait, with, “I could say the same of yours if I let myself sink to that level …”’ Clearly, the latter quotes are signs of friction, i.e. the presence of an affective conflict.
CONFLICT MANAGEMENT IN OPEN SOURCE COMMUNITIES

The next section identifies and discusses a number of mechanisms that perform a role in the management of conflicts in open source communities. These mechanisms have a number of sources. Third-party intervention has been suggested in literature as performing a key role in the management of many conflicts. The other mechanisms have been identified from interviews with respondents and from secondary literature. They are: modularity; parallel development lines; and the exit option.

Third-party intervention in OS communities

One frequently mentioned mechanism to resolve conflicts is by third-party intervention (e.g. Elangovan 1998; Nugent 2002; Pruitt 1998). Two types of third-party intervention are relevant here, namely mediation and arbitration.\(^9\) 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). In the next pages the mediating and arbitrating roles of PMC’s in Apache and Linus Torvalds are discussed.

First, let us consider the role of the Project Management Committees (PMCs) in Apache. According to the Apache website,\(^9\) Apache is divided into parent projects. Each of these parent projects has a PMC, which is responsible for the technical direction of the project. As a management committee responsible for the technical direction it would be reasonable to assume that they take on a mediating or even arbitrating role when conflicts occur, especially when it involves technical issues. One of the co-founders of the Apache Software Foundation and current board members of that same foundation, argues: ‘Nowadays, the PMCs are too far away from the actual coding. The PMCs do little …’ PMCs are argued to do hardly anything. Two other members of the foundation made the same claim, by saying that the PMCs were fairly inactive. Thus, one can wonder whether PMCs actually have a role in the management of conflicts in the Apache community.

Linus Torvalds is the creator of the Linux kernel 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 2002). 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 be missing in current research is the possibility that he is a mediator or an arbitrator.

Let us consider an example of a conflict, which centred on a specific part of the Linux kernel, namely Virtual Memory. There still are two alternatives: one is maintained by Rik van Riel and the other by Andrea Arcangeli. The conflict manifested itself for the first time in May 2000. Since then, the conflicting parties have had many clashes. Zack Brown describes one of these clashes. He writes:

There are many more of these examples of the conflict, which can be observed on the Linux kernel mailing list in the period from May 2000 to December 2001. 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.’\(^10\)

These and other statements do seem to contradict 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.

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, suggests that Linus Torvalds is an arbitrator, because Andrea’s virtual memory has always been the preferred choice of Linus Torvalds. There are, however, also many reasons that might contradict this idea. The most obvious reason is the fact that the conflict between Andrea and Rik has continued a very long time. Linus made his opinion heard on many occasions and in many instances he expressed his preference for Andrea’s system, but there is no evidence that suggests that this contributed to resolution of the conflict. Based on this example, the presence of Linus Torvalds appears to have had no significant influence on the resolution of the conflict. However, more detailed research and a comparison with other examples might prove otherwise.
This section provided some first discussions about the role project leaders and communities perform in the management of conflicts. First, empirical observations suggest that the role of Linus Torvalds in Linux and the PMCs in Apache is rather limited for the following reasons: (a) many conflicts appear to be solved without the interference of a third party; and (b) a first observation of the mailing lists has provided no significant evidence that supports the claim that conflicts were resolved because Linus Torvalds or the PMC got involved.

Modularity

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 a Linux programmer:

> Of course, Linux software is very complex, but on the other hand of course it is not. The answer is to divide and conquer. When you have a complex piece of software, you cut it into ten pieces and if you manage to provide them with good interfaces then you only need to understand the separate pieces.12

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 2002, p. 28). Modularity results in fewer conflicts for two reasons. It increases independence between programmers and hence reduces the amount of conflicts (e.g. Jehn 1995). Programmers work on specific and relatively isolated parts of the software. Without modularity, changes made in one part of the software would likely have the effect that ‘something else does not work anymore.’ Therefore, programmers have to make many decisions collectively and would be much more likely to frustrate each other’s development efforts. Modularity decreases the dependency and hence the chance that a conflict occurs.

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

Parallel development lines

Another mechanism is the presence of multiple or parallel development lines. The Editor-in-Chief of Linux Journal explains: ‘It seems like there are often two different ways of tackling a problem and people try both.’ Theoretically, the presence of parallel development lines enables programmers 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 of parallel development lines 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.’14

In many cases it is evident which of the development lines is the better one and 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 their development. 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 communities as clear and obvious as the diversity between non-technical contributors and highly skilled and trained software programmers. 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.15 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 ensures that
conflicts come to the surface less. Programmers and users who are less skilled and favour a stable software package will download the stable version of the software and are ensured that their interests are at least partly satisfied. Highly skilled programmers, on the other hand, will be inclined to participate in the development of the experimental version of the software. Now, they are much more challenged to use their skills and test new ideas.

The exit option

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 to 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, avoids the rise of conflicts, as it communicates 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 …’ (p. 76). Given the fact that the voice option will lead to conflicts, this statement implies that the exit option will diminish the rise of conflicts, because it is an effective way to substitute voice. When people have the possibility to exit, they will use their voice less and will thus create fewer conflicts. To identify the presence of an exit option will result in the conclusion that the voice option is used less, which makes it an effective means to avoid certain conflicts within a team.

The exit option is institutionalized in many different ways in open source communities. The most dramatic 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.’16 Most forks start out as parallel development lines. The difference, however, is that in a fork the conflicting parties are no longer in the same community and that the conflict is no longer a conflict within a community but has transformed into a conflict between communities.

Much research considers forks to be negative. Narduzzo and Rossi (2003, p. 29) for example state: ‘These forks are an expression of a coordination failure …’ Kuwabara (2000, p. 48) has a similar stance when he writes: ‘Forking … and other behaviors become taboos.’ In other words, both authors argue that a fork is bad and should be avoided. They feel that forking is bad because it destructs value: it gives rise to two communities in which software is developed that could become incompatible. However, programmers themselves do recognize that sometimes forking is necessary: ‘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!’17

To summarize: forking is a way to avoid current and future conflicts. Furthermore, it can be taken as an indication that programmers who worked together in a community have experienced conflicts, which they were unable to resolve. The interesting research challenge would be to understand when and why forks in open source communities have occurred. One finding could for instance be that a fork only occurs when the conflict actually threatens the continuity of a community. If so, one could argue that the fork is a last resort to save the community by allowing one or more conflicting parties to use the exit option and to leave the community. At the same time the fork allows the parties to prove that their arguments were indeed right.

AN EXAMPLE: COVALENT AND THE APACHE COMMUNITY

Covalent, a California based company, has strategically deployed the mechanism of parallel development lines. It did so by creating its own Apache software development line, which enables it to develop the software independently from the Apache community. In its version, Covalent can optimize and reconfigure any part of the official Apache tree and ensure that its version best meets its customers’ requirements. Even if these interests are less relevant for other members of the community they can implement changes to their own in-house development line. This way, Covalent shields customers from the community. ‘The ASF [Apache Software Foundation] may release four new versions in one week … our customers don’t notice it though, because we isolate them completely from the ASF.’18 For Covalent, the biggest advantage of disconnecting the development lines is that it no longer needs to pressure and it has no direct dealings with other programmers in the community. ‘We just don’t care if the ASF releases a new version or not.’19

The dual development lines enable Covalent to disconnect its development efforts from the Apache development line. Covalent’s efforts focus primarily on
creating a product that better meets the wishes of its customers. Therefore, it runs a lot of tests and does a lot of bug fixing. These bug fixes are added in Covalent’s version immediately, but they are also sent to the Apache community. In most cases, the Apache programmers do add the fix in their version as well. The commercial development line of Covalent might resemble a fork, but it is different. The difference is that constant synchronization between the two lines prevents the rise of irreconcilable differences and it allows programmers in the Apache community to benefit from Covalent’s involvement. Thus, in a way the programmers in Covalent are actually members of the Apache community, as they contribute source code. The commercial development line allows Covalent to maintain a version in which mistakes are not tolerated, in which deadlines for new releases are met and in which conflicts with other programmers in the Apache community come to surface less.

The mechanism of parallel development lines method also has drawbacks. The most obvious is the continuous need to integrate the improvements made by Covalent with the official Apache version and vice versa. Covalent must continuously monitor the developments in the Apache community. The community might find and fix a bug after Covalent downloaded a version of the software. In that case, Covalent must integrate the bug fix into its own version as well. In other words, the commercial development line does raise the costs of software development. On the other hand, it does allow Covalent to capitalize on the development efforts of many other programmers.

CONCLUDING REMARKS

The basic idea that underlies this paper is that companies that want to create their business model based on OS software have to understand the dynamics of OS communities and their software development process. Understanding the dynamics and using the mechanisms that are currently present in OS communities, is likely to result in a more sustainable business model. This paper focuses on one specific issue that most companies are bound to run into when adopting OS software, which is the presence of conflicts between their programmers and voluntary programmers and the obvious need to manage these conflicts.

The paper identifies and discusses four mechanisms that are believed to perform a role in the management of conflicts. They are: third-party intervention; modularity; parallel development lines; and the exit option. Limitation, however, is that this paper is based on a highly explorative study, which does not allow the conclusions to be generalized. Further research is therefore suggested. Such research should aim to contribute to a better understanding of the management of conflicts in open source communities. Quantitative empirical research could be conducted to verify whether the identified mechanisms are present and to understand their role in the management of conflicts. Such research could possibly lead to the conclusion that a mechanism like parallel development lines, will bring the number of task conflicts in a community below the optimal level and will thus have a negative effect on the performance of that community.

Quantitative research on conflict management in open source can provide a valuable contribution to our current understanding of open source communities. Furthermore, it can provide valuable lessons to companies that want to get involved in open source communities. And finally, it will add to the body of empirical research that currently exists on conflict management.

Notes
3. Cited from an interview.
4. According to figures that are publicly available on the Internet Apache has a market share of almost 70%.
5. Translated from Dutch.
6. Translated from Dutch.
8. 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 analysed 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 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 analysed, but is considered to be a specific form of arbitration.
10. Taken from the Internet: http://kt.zork.net/kernel-traffic/kt20010112_102.html#10 (last visited 14 April 2003).
12. Translated from Dutch.
13. Translated from an interview in Dutch. The interview was held with one of the two maintainers of the Lilypond software.
14. Cited from an interview with the chief editor of Linux Journal.
15. Based on an interview with the creator and maintain-er of the Python programming language.
17. Cited in an interview with the former project leader of Debian, translated from Dutch.
18. Cited from an interview with a board member from the ASF (translated from Dutch). The ASF is short for the Apache Software Foundation, which is the main body in the Apache software community.
19. Cited from an interview with a board member from the ASF (translated from Dutch).

References


