Almost by definition, Open Source Software (OSS) offers an incentive to elaborate and adapt source code. One would expect this in-built opportunity to diverge to lead to incompatible strains of software, and, consequently to a clamour for standardisation. However, this is only partly the case. Why? Which other coordinative mechanisms are at work apart from the standards initiatives? From standards literature we distil four categories of coordinative mechanisms, and illustrate their relevance with OSS examples. In the concluding section we re-address the complementary relation between the four categories and committee standardisation.

“What if, when the software doesn't work or isn't powerful enough, you can have it improved or even fix it yourself? What if the software was still maintained even if the company that produced it went out of business?” (Perens, 1999, p. 171)

Perens refers to the possibilities of Open Source Software (OSS) to adapt and improve source code to specific user and developer requirements. Open source code means that anyone with knowledge of software programming can understand how OS software works. Moreover, the user licenses that govern OSS development explicitly allow modification and redistribution of the modified source code. The majority of OSS is freely available, easily accessible and downloadable from the websites of open source communities. Open source communities primarily consist of hobbyists and many of them are very large. For example, the Apache community roughly comprises 630 contributors. Thereof, about ninety are members and belong to the core-developer group. Together these communities develop and maintain a variety of mature software programs (e.g. Linux and Apache).

The openness of the source code and the licenses, present an opportunity for anyone with access to the Internet to modify the source code and start a new branch thereof. Let us consider the number of people that might do so. Imagine a thousand people download a software program – a much too modest figure. When taking into account that 10% of them have a background in software engineering and another 10% actually feels the need to modify the software (e.g. they found a bug), this still means at least ten modifications to the original software. (Van Wendel de Joode et al., 2003)

There is no entrance fee for participating in OSS development. In most cases, one does not even need to register to download software and modify its source code. Low entry barriers ease the emergence of diversity in OSS (Franck & Jungwirth, 2002; Sharma et al., 2002). The absence of any significant entry barriers - apart from engineering expertise - also explains the variety among those actively involved in OSS development:

“(T)here are many different types of contributors to open-source projects: organizations and individual initiators and those who help with other people's projects; hobbyists and professionals.” (Markus et al., 2000, p. 15)
Variety within the OSS community, in turn, brings along different requirements as to what a particular type of software is supposed to do. With both motive and opportunity to modify code, one would expect many branches (‘forks’) and variants in open source software. This diversity more likely leads to incompatibility.

As Kogut & Metiu (2001, p.257) note, "[t]he danger of open-source development is the potential for fragmenting the design into competing versions." The classic example of fragmentation is UNIX, a multi-user operating system and an open source initiative avant la lettre. UNIX was a de facto standard in the late 1970s. However, different UNIX variants developed, which fragmented the market. Incompatibility among OSS variants is currently still an issue, as box 1 illustrates.

**Box 1: Standards initiatives for Linux**

Standards initiatives have taken place to increase compatibility between the different Linux distributions, which have recently emerged. Firstly, although the market had its favourite, an effort was made to standardize Linux under the auspices of the Linux Standards Base (LSB) - which is the name of the organization as well as the resulting standard. The aim of the organization is "to develop and promote a set of standards that will increase compatibility among Linux distributions and enable software applications to run on any compliant system..." The standard (LSB), which was first released in June 2001 “provides a way to ensure behavioural compatibility across Linux distributions and version releases. An application written to the standard will function the same across all LSB certified platforms...”

Secondly, to assure compliance with the standard, the Free Standards Group, a non-profit group supported by industry, which advocates standards for open source technologies, contracted The Open Group, an organization specialized in interoperability certification, to manage LSB Certification.

Thirdly, a group of Linux providers formed an industry consortium called UnitedLinux. The consortium took the Linux Standards Base as the starting point for its common operating system UnitedLinux. Version 1.0 was released in November 2002. This initiative, too, aimed at increasing convergence. It “streamline[s] Linux distributions in the marketplace.”

Lastly, technical compatibility in OSS also benefits by inter-organisational coordination and organisations such as the Free Software Foundation (FSF). The latter frequently plays a mediating role between open source communities, as the following quote illustrates. “We had success with this role with the GNU Compiler Collection (GCC). A couple of years ago the FSF had problems maintaining the GCC project, stewardship of the project was not working. A large group of developers with the support of Cygnus, which is now part of Red Hat, created a fork, which was called EGCS. Instead of competing with this fork, the FSF decided to talk to this group and negotiated an agreement in 1999. In that year both projects merged and became GCC again.” (Bradley Kuhn, vice-president of the FSF)

Summarizing, in the case of Linux coordination and convergence is addressed through standardisation and certification of standards-compliance, and cooperation and mediation between groups.

Having said this, potential incompatibility has not prevented OSS to attain a dominant market share in certain areas (e.g. 90% of all e-mail programs are based on Sendmail, and Apache has a 65% share in the webserver market), or a very promising position in the market (e.g. the Linux kernel is supported by a number of large and

---

5 The US government, which had many defence contractors writing software in UNIX and using slightly different systems, stepped in to create a common UNIX version called POSIX. IEEE took this up for standardisation in 1983, but it did not prevent further divergence. “POSIX promised to heal the split between the two main UNIX camps: Berkeley UNIX and AT&T's System V. Unfortunately, a new split appeared when a group of vendors (IBM, DEC, Hewlett-Packard and others) set up a consortium called OSF, the Open Systems Foundation [whose intent was to base their work on POSIX]. In answer, AT&T and its followers wanted to protect their privileged position by setting up their own consortium, UI or UNIX International to do the same thing as OSF.” ([www.cs.cansius.edu/UNIX/history.html](http://www.cs.cansius.edu/UNIX/history.html))

At present there are more than fifty UNIX flavours ([www.ugu.com](http://www.ugu.com), 15 May 2003). One of these is the Single UNIX Specification Version 3, a standards effort that combines “industry-led efforts and the formal standardization activities into a single initiative... with the resulting set of specifications being approved as IEEE Std 1003.1-2001 (POSIX), ISO/IEC 9945:2002 (parts 1 to 4), and The Open Group Base Specifications, Issue 6.” ([www.unix-systems.org/version3/online.html](http://www.unix-systems.org/version3/online.html), May 2003.)

6 [www.linuxbase.org](http://www.linuxbase.org), May 15, 2003


8 [www.unitedlinux.com](http://www.unitedlinux.com)

highly respected software houses such as IBM). Market success implies coherence and compatibility. The degree of diversity in OS software seems to be tempered by coordination efforts other than - and in addition to - standardization. This brings us to our main research questions:10

How do the coordination mechanisms other than standardisation work in OSS development? What is their significance for standards policy?

The paper is organized as follows. In standards literature coordination is a theme discussed in some detail. We first scan this body of theory to acquire ideas about what mechanisms to look for. We arrive at four categories of coordination, apart from committee standardization. These four categories are used to structure our discussion of coordination in OSS development, a discussion largely based on interview data described in the methodological section. The section closes with a comparison of the mechanisms used in the development of Apache and Linux. Finally, we readdress the relation between committee standardization and the other four categories of coordination, and make recommendations for further research.

Theories of coordination

Some form of coordination is required to maintain consistency, prevent divergence and/or achieve technical convergence and, preferably, compatibility and interoperability among software. Without wanting to suggest a comprehensive overview, we will introduce a few salient coordination mechanisms discussed in standardisation literature.

Several classic studies on coordination take an economic perspective. Therein two types of coordination mechanisms are compared (e.g., Katz & Shapiro, 1985; Farrell & Saloner, 1988): committee standardisation and coordination by the market. IT-company standardisers - and some scholars - may feel somewhat uncomfortable with this categorization if they view committee standardisation as a market process. There are other objections as well (see the discussion on network externalities). However, we use the distinction here to organise some main theoretical concepts in a recognisable way. The concepts will later be used as a stepping-stone for identifying coordination mechanisms other than committee standardisation in OSS development.

Standardisation

Standardisation is a response to technical and actor complexity, as Schmidt & Werle (1991, 1998) argue. From the technical standpoint, the need for coordination follows from the technical interrelatedness of components and products, that is, from the ‘network character of many technical artefacts’ (Schmidt & Werle, 1991, p.2). Complex technical interrelatedness usually correlates positively with actor complexity. That is, the more complex the system is in terms of number of different components, the higher and more diverse the number of actors involved. The more multiple actor interdependencies, the higher the need to coordinate market strategies and define common technical specifications for product and technology development. Committee standards are such specifications. They serve to create technical compatibility among products and services, and aim to direct the actions and orientation of other market players. Standards “help coordinating the actions of those who design, produce or use technical components as parts of more comprehensive technical systems.” (Schmidt & Werle, 1991, p.6)

As discussed in the introduction, the standards efforts in OSS indicate that standardisation is a highly relevant coordination mechanism. In addition, there are OSS-oriented initiatives that complement committee standardisation. Our research question focuses on the latter coordination mechanisms. At stake are usually - compared to standardisation - more implicit types of coordination.

10 See the paper of T.M. Egyedi presented at the Free/Libre and Open Source Software (FLOSS) workshop, organised by the International Institute of Infonomics, University of Maastricht October 14, 2002, European Commission, Brussels, Belgium. (www.infonomics.nl/FLOSS/workshop/index.htm)

It is important to note here that this research does not address the question what the motives of firms and volunteers are to participate in the community. A lot of research has already focused on this question (e.g. Markus et al., 2000; McGowan, 2001; Lerner & Tirole, 2002; Von Hippel & Von Krogh, 2002). Instead, the challenge is to understand processes in the communities given the active involvement of firms and individuals.
Market coordination

In economic articles on standardisation, two concepts related to market coordination are frequently mentioned, i.e.: the “bandwagon mechanism” and “positive network externalities”. Farrell & Saloner (1988, p.236) define the bandwagon mechanism as follows:

"If an important agent makes a unilateral public commitment to one standard, others then know that if they follow that lead, they will be compatible at least with the first mover, and plausibly also with later movers.”

The “bandwagon mechanism” is a descriptive concept. It captures a leadership-induced response by the marketplace in situations where there is technical choice. Technical convergence is a by-product of this response.

The bandwagon mechanism has a socio-cognitive side to it as well as an economic side. Katz & Shapiro (1985, p. 439) speak of psychological bandwagon mechanisms. Other group dynamic processes could be added (e.g. reducing cognitive dissonance by rationalising one’s choice based on indicators of quality and technical excellence).

We expect that the socio-cognitive side may prove important with regard to ‘coordination in the market of OSS’.

As yet there is little insight in how the OSS market works. Prognoses are uncertain. This makes marketing issues such as ‘reputation’ and consumer expectations all the more important.

The notion of positive network externalities or consumption externalities is of an analytic kind. Katz & Shapiro (1985, p. 424) circumscribe it as follows: “There are many products for which the utility that a user derives from consumption of the good increases with the number of other agents consuming the good.” Specifying their idea, they identify three main sources of network externalities (Katz & Shapiro, 1985, p. 424):

• the direct physical effect of number of purchasers on the quality of the product (e.g. the number of connected households increases the value of a connection to the telephone network)
• the indirect effects (e.g. the more computers that are sold of a certain kind, the more software will be developed for this type of computer)
• the quality and availability of post purchase service for the good depend on the experience and size of the service network, which may in turn vary with number of units of the good that have been sold (e.g. service network for popular car brands is denser and possibly better since same set of servicing skills are needed).

The notion of network externality provides a rationale for coordinative responses, and, in extremo, for the emergence of monopolies. It provides a rationale for coordination but is itself not a mechanism of coordination. Let us explain. In the introduction to section 2, we hinted that a categorisation of theoretical notions based on the market vs. standards committees division is problematic. This is because ‘network externalities’ are always treated under the headings of ‘the market’. However, the ‘network externalities’ rationale explains committee standardisation as well as other ‘market-based’ responses to coordination needs. For example, in complex markets the externalities of compatibility are inherently high. Where this is the case, they create a pull towards compatibility, as do other, market-bound rationales like reaping the fruits from ‘economies of scale’. In both cases, the forces at work trigger coordination mechanisms.

---

11 Game theory is frequently used to model coordination by committees and markets (e.g. battle-of-the-sexes game, war of attrition, grab-the-dollar game; Farell & Saloner, 1998). The assumptions needed to reduce the complexity of the coordination problem (e.g. that players have a choice between two incompatible new technologies or systems which are championed by different companies) make game theory less suitable for understanding OSS development. However, the motives mentioned in this literature for choosing one standard above others may very well be applicable to choice between OSS software (e.g. technical superiority, vested interests, expectations, and company reputation).

12 Liebowitz & Margolis specify ‘positive consumption externalities’ by distinguishing two components to consumer value: the autarky value, which is “the value generated by the product even if there are no other users”, and the synchronization value, the additional value “derived from being able to interact with other users of the product. (...) [I]t is this latter value that is the essence of network effects.” (Liebowitz & Margolis, 1998, p.1)

13 Product information is also more easily available for popular brands (Katz & Shapiro, 1985, footnote 1).

14 For example, McGowan & Lemley (1998) describe how in respect to goods like operating systems, where network effects are strong, this rationale seems to acquire the nature of an economic necessity. Coordination then results in the emergence of monopolies or oligopolies.
Are the above concepts applicable to OSS development? The open source movement is a radical kind of joint, distributed, and networked effort. A network, like a market, is a way to coordinate and cooperate activities. However, in markets a sharp distinction exists between contributors (suppliers) and users (buyers); whereas, it is characteristic for a network that all participants make a contribution (Van Wegberg & Berends, 2000a/b, p.5). This difference means that other coordination mechanisms may be required in OSS settings.

Other coordinative efforts
Under the headings of compatibility-enhancing strategies, Egyedi (2000) discusses a set of measures, which Sun Microsystems took with regard to Java technology. These measures have coordinative implications. They are listed in table 1. At stake are coordination mechanisms. Some are typified as coercive strategies. Their coordinative effect depends on the implicit guidance provided to the mechanisms and their voluntary compliance to the mechanisms (e.g. instructional books and standardization, respectively). These depend on compatibility pull. 15

Other strategies more forcefully impose compatibility (compatibility push). Typically, coordination is then enforced by administrative requirements (e.g. membership registration) and legal means (e.g. licenses and contracts). In Sun’s case, instruments such as licensing, test suites and the Java Community Process were used to safeguard the compatible development of Java.

<table>
<thead>
<tr>
<th>Coercive strategies</th>
<th>Forceful strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Source Code</td>
<td>Community Process</td>
</tr>
<tr>
<td>Instructional books</td>
<td>Participation Agreement</td>
</tr>
<tr>
<td>Certified training programs</td>
<td>Technology License and Distribution Agreement</td>
</tr>
<tr>
<td>Distribution of the Software Development Kit</td>
<td>Community Source Licensing model</td>
</tr>
<tr>
<td>ANSI/JTC1 Standardization:</td>
<td>Reference Implementations</td>
</tr>
<tr>
<td>JTC1 PAS process</td>
<td>Test suites</td>
</tr>
<tr>
<td>ECMA/JTC1 Fast Track procedure</td>
<td>Compatibility logo</td>
</tr>
</tbody>
</table>

Table 1: Compatibility-enhancing coordinative strategies used by Sun for Java (adapted from Egyedi, 2000).

To condense the different examples into a more easily surveyable list, we define a category of regulatory coordination in which company rules and government regulation enforce coordinative behaviour. Typical means to do so include contracts, licenses, intellectual property rights and participation agreements (e.g. membership registration).

For example, in March 1996 the Technology License and Distribution Agreement for Java came into effect between Sun Microsystems and Microsoft 16. To illustrate what such an agreement entails, the first part of the agreement addresses the need for compatibility among Java-based products and value of the Java compatibility logo. It says that Sun expects Microsoft's products to incorporate the latest updates of the Java platform. To test their compatibility, Sun offers Microsoft its test suites free of costs. If the products pass the tests, Microsoft may use the Java-Compatible logo - which has become associated with much goodwill - for marketing purposes. See Egyedi (2000) for more detail.

Other coordinative strategies mentioned in the table, like instructional books on Java, participation agreement for the Community Process, certified Java training programs, reference implementations, testing and distribution of the Java Software Development Kit, we captured under the headings of operational coordination mechanisms. They address the operational part of technology development and use. Coordination here is technology-specific.

15 ‘Compatibility push’ and ‘compatibility pull’ are defined as (Egyedi, 2002c): Compatibility pull: The idea is that in some situations the benefits of interoperability are so evident, that it creates an inductive force. The parties will voluntarily conform to agreements that increase interoperability or actively pursue it by other means. Compatibility push: Conversely, other situations sometimes enforce interoperability in one way or the other. For example, market monopolies or legislation. One feels pressed to buy certain products or conform to a certain standard.

16 Usually, licenses for commercial use are confidential. In the case of Microsoft's license to use Sun's Java, the confidentiality clause was broken because the two parties were involved in a lawsuit from 1997-2001.
Again, using the Java case as an example, the instructional books which Sun and other authors write on Java and the training programs that lead to certified Java developers contribute to a co-ordinated development of the Java trajectory. Those who follow the program will learn the tricks of the trade (heuristics), are introduced to the Java philosophy, etc., - that is, they become familiar with the ‘Java paradigm’. A shared outlook increases the likelihood of a same-faced outcome. These activities support the development of a Java practitioner community.

However, they do not guarantee compatible Java implementations. In this respect, a more effective compatibility strategy is the distribution of the Java Software Development Kit (SDK, formerly JDK). The SDK contains ‘a full suite of development tools (…) to write, compile, debug, and run applications and applets’ (e.g. implement new APIs) and the Java runtime environment (i.e. the execution environment needed to run applets and applications). Use of the SDK narrows down the number of possible programs to those that run on ‘standard’ Java platforms. (Egyedi, 2000)

Farrell & Saloner (1988) mention in passing another coordination mechanism: coordination by authority. This is similar to what Schmidt & Werle (1991) address as ‘coordination by hierarchy’. Coordination by authority occurs “if one player is nominated in advance to choose [among alternatives], and others then defer to him” (e.g. gatekeepers). We stretch their definition in order also to include coordination mechanisms based on informal authority. (Informal) authority can be based on a variety of resources: technical expertise, economic success, formal position in the organisation, etc.

For example, in December 1998 Sun issued a first version of The Java Community Process (sm) Program Manual: The formal procedures for using the Java Specification development process (version 1.0). It was heavily criticised among other things because in the proposed procedures Sun would still veto decision regarding the eligibility of specification requests. It was seen as a ‘gated’ community process.

<table>
<thead>
<tr>
<th>Categories of coordination mechanisms</th>
</tr>
</thead>
<tbody>
<tr>
<td>committee standardisation</td>
</tr>
<tr>
<td>market / socio-economic coordination</td>
</tr>
<tr>
<td>regulatory coordination</td>
</tr>
<tr>
<td>operational coordination</td>
</tr>
<tr>
<td>coordination by authority</td>
</tr>
</tbody>
</table>

Table 2: Five main categories of coordination relevant to compatibility issues\(^{17}\).

Summary of conceptual outlook on OSS

Below, we use the following categories to explore coordination mechanisms in OSS development:

- **market or socio-economic coordination mechanisms.** As argued before, we restrict ourselves to bandwagon phenomena as triggered by e.g. reputation, trademark, consumer expectations, market share as a sign of product quality, records of activity as an indication of technical excellence, purely psychological bandwagon effects and other group dynamic processes.
- **regulatory coordination mechanisms** such as contracts, intellectual property rights (e.g. licenses), participation agreements (e.g. membership registration).
- **operational coordinative mechanisms** like software support tools, training, educational books, reference implementations, testing, etc.
- **coordination by authority** by means of gatekeepers, informal or formal hierarchy, etc.

Coordination mechanisms in open source communities

Data has been collected that allowed us to identify the mechanism that lead to coordination in open source communities. This data is based on content analysis of documents, websites and mailing lists, a study of secondary literature, and, in particular, on face-to-face (semi-structured) interviews with experts actively involved in OSS development. The interviews took place from October 2001 to August 2002. 47 respondents from the Netherlands,

\(^{17}\) NB: The categories emphasize different features of coordination. They are largely, but not fully, mutually exclusive.
the US and Germany were interviewed. 39 of the respondents were active software developers in a wide range of communities and 8 performed functions other than the development of software.

This article will focus on two communities in particular, namely Apache and Linux. Both names refer to the software as well as the communities that develop and maintain the software. The Apache software is a web server. The primary task of a web server is to host content, like websites and databases. Rob McCool laid the foundation for the Apache web server, while working at the National Center for Supercomputing Applications (NCSA). A small group of geographically dispersed people enhanced the software. Growth of the software and the community continued and currently the Apache web server hosts more than 65% of the active Internet websites. Linux is an operating system. Linus Torvalds started development of the Linux kernel in 1991. By placing the source code on the Internet Linus was able to attract many people who together have been able to create an operating system that is said to be the only real threat to the hegemony of Microsoft Window (Wayner, 2000).

The next sections will address the coordination mechanisms in both the Apache and the Linux community.

**Bandwagon mechanism**

In the interviews there was one thing that struck us, namely the idea that popular open source projects have a tendency to become even more popular. Consider in this light the statement from one of the respondents: “It is like a chain reaction, popularity will lead to more users, more users will lead to more developers, to more applications and thus to more users, etc.” At work is a self-reinforcing, evolutionary selection mechanism, which creates a process similar to the bandwagon effect. Below we discuss two of the many causes of the bandwagon effect: reputation of developers and level of activity.

One of the most cited motives for people to get involved in open source software development is the enhanced **reputation** that one receives when making high-level quality contributions to an open source project [e.g. Hippel, 2001 p.265; Markus, 2000 p.192; Benkler, 2002 p.271; McGowan, 2001 p.276; O'Mahony, 2003 p.362; Ljungberg, 2000 p.270; Selz, 1999 p.240]. If we assume that reputation can only be gained through high-level contributions, reputation becomes a sign of quality and developer expertise. Developers with a high reputation are likely to be good developers.

One effect of reputation is that to earn a good reputation developers will choose open source projects that provide them with a large audience (Lerner and Tirole, 2002). Popular open source projects consist of many developers and users and will thus be attractive projects to start developing on. In other words, popular projects are likely to attract developers that seek to gain a high reputation. The presence of other developers with a high reputation will also cause more developers to choose that community. The former project leader of the Debian community explains that “Reputation does help: when you see a good name, you will take a look.”

To summarize: the reputation of a developer is a sign for other developers to choose the same open source project. This also explains why popular projects are usually of high quality. Obviously, a community that consists of highly skilled developers is more likely to produce software that is of high quality.

A second mechanism that reinforces the bandwagon effect is the **level of activity** in the community. Many websites list statistics that indicate how active projects are. One such site is Freshmeat, which contains a so-called ‘popularity index’. Another site is SourceForge, which hosts a large number of open source projects. They use a number of statistics to measure the activity in projects, e.g. activity and the number of downloads. These statistics affect people’s choices. One respondent explains:

> “SourceForge has information on activity as well: if there are twenty different versions of a library for a certain purpose, and one has been downloaded 10,000 times and another one 20 times, it is clear what you choose first.”

Thus, the level of activity is a sign for developers to get involved in a certain project. This leads to positive feedback: the new developer will also download software and by doing so will automatically increase the level of activity. Why do developers choose projects that have a high level of activity? The reasons are quite straightforward. A higher level of activity implies that the newcomer will receive a quick response to the questions asked. It also signals the presence of higher quality software, as problems are sooner discovered and fixed.

---

18 The index is based on the number of subscriptions, URL hits and record hits. (software.freshmeat.net/stats/p.popularity, May 21, 2003)

19 sourceforge.net/top/ (May 21, 2003).
Regulatory coordination mechanisms

Law is an important source of coordination in open source communities. There are three sections of the law that are important in this respect. One part consists of intellectual property rights and especially copyrights, as they lay the foundation of the open source licenses (e.g. Markus et al., 2000; Wayner, 2000; Dalle and Jullien, 2003; McGowan, 2002; Perens, 1999; Bonacorsi and Rossi, 2003). Arguably, certain open source licenses promote convergence more than others. Another part consists of trademarks. Most open source communities have trademarked the name of their software, which has a coordinating effect. Finally, there is contractual law. In a number of communities developers have created contracts, which newcomers have to sign before they are allowed to contribute to the source code. Some of these contracts have a coordinating effect.

As noted earlier, there are a great number of different open source licenses. The Open Source Initiative hosts a website where they collect and provide copies of licenses they consider to be open source licenses. Currently they list 43 licenses. These have in common that the source code is open and that others are allowed to change the code. Therefore, each of these licenses essentially facilitates further divergence. Still, certain licenses do so more than others. To illustrate this we analyse two of the most popular licenses in more detail: the General Public License (GPL) and the Berkeley Software Distribution (BSD). Most other open source licenses are elaborations of one of them (Van Wendel de Joode et al., 2003).

<table>
<thead>
<tr>
<th>License</th>
<th>Can be mixed with non-OSS</th>
<th>Proprietary distribution is possible</th>
<th>Can be re-licensed by anyone</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPL</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>BSD</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 3: A summary of two open source licenses.

In table 3 the two licenses are compared on three aspects. The first is whether or not the software licensed can be mixed with non-open source software. As the table shows, software licensed under the GPL cannot be mixed with non-OSS. Reason being that the license explicitly states that software combined with GPL software becomes licensed under the GPL as well. Software licensed under the BSD can be mixed with non-open source software. The second characteristic is proprietary distribution. A proprietary distribution of software means that the source code of the software is closed. Thus, as table 3 indicates, GPL software cannot be distributed in a proprietary version, while software under the BSD license can. The third characteristic is re-licensing. Both licenses forbid others to re-license the software.

How do these licenses affect the level of convergence? The GPL stimulates more convergence than other licenses: “If you want to get the source code for modifications back from people who make them, apply a license that mandates this. The GPL... would be good choices.” (Perens, 1999, p. 185) However, the GPL does not require people to send back their modifications to the original author of the software. It only obliges people distributing GPL software to disclose the source code and the modifications made to the source code. Still, the GPL does support some level of convergence as others have the possibility to integrate modifications into the original software. This is different from the BSD license. Software licensed under the BSD can be taken private, meaning that the modified source code does not need to be made available. Therefore, software licensed under the BSD is more prone to divergence.

Trademarks are symbols related to a specific product. They are used to differentiate the product from competing products. For example, Linux, Apache and Debian have trademarked their names for this purpose (O'Mahony, 2003). On a number of occasions the Apache Software Foundation had to protect their trademark against others who called their product, for example, WinApache. Trademarks diminish divergence for two reasons. Firstly, well-known trademarks like Apache and Linux attract new developers, thereby reinforcing the bandwagon effect. Secondly, trademarks makes it less likely that developers create a different version of, for instance, Apache. Since

---

20 www.opensource.org/licenses/ (May 2003).
21 According to the Freshmeat statistics the GPL is by far the most used license. No less then 65 percent of the projects uses the GPL. The second popular license is the Lesser GPL (LGPL), which is based on the GPL and differs in only one aspect from the GPL. The third popular license is the BSD license, which is used in 5,3% of the projects. The Apache license is used in only 1 percent of the projects that are analysed by Freshmeat. (software.freshmeat.net/stats/, May 21, 2003).
22 The table has been adapted from Perens (1999), who provides a more elaborated discussion on open source licenses.
the Apache trademark is well known, it will be difficult to start a project based on the Apache software that achieves the same level of popularity using a name that in no way resembles that of ‘Apache’.

Apache requires developers to sign a participation agreement before allowing them to include source code directly into already existing software. The primary reason is to prevent them from contributing software that includes material copyrighted by a company. Having people sign a participation agreement may increase convergence, but it may also create more divergence.

Agreements stimulate convergence because the agreements function as a boundary: people have to make an effort before they can become a part of the Debian or Apache community. Boundaries in general are a way to increase the chances that people get to know each other, which will reduce uncertainty and has a coordinating effect (e.g. Ostrom, 1990). People will feel part of the community, which makes adherence to one version more likely. A negative side of such agreements is that people first need to make the effort of signing the agreement and thus crossing the boundary. If people are not willing to make this effort, agreements may increase divergence. Although, they cannot upload source code they can still download the software and modify it. As they are not allowed to upload the software, chances are high that they will make the modified versions available on another website and by doing so create divergence.

Whether or not people are willing to cross the boundary and become a recognized part of the community depends on the person, the costs incurred by becoming a member of the community (e.g. a reduction of costs), and on the popularity of the community. As mentioned before, a popular community is likely to attract developers. If they are popular enough, people are more likely to cross the boundary and sign the agreement. In this situation the presence of boundaries is likely to stimulate convergence rather than divergence.

Operational coordinating mechanisms
The third group of coordinating mechanisms are operational coordinating mechanisms. Developers in open source communities use a great number of tools to support their activities. These tools and especially the heuristics used therein focus and structure the work within these practitioner communities. The interviews suggest that two of these tools are particularly relevant: the concurrent versioning system (CVS) and the manuals.

One of the most important operational coordinating mechanisms is the CVS. The CVS is a client-server repository. It is an automated system that allows remote access to source code and, according to the firm that develops and markets the CVS, it enables multiple developers to work on the same version of the source code simultaneously. A board member of the ASF describes the software development process in the CVS roughly as follows.

A typical software development process in the CVS starts with a first version of the software. Developers in the community download this version and try to understand how it works. Once they understand the software they are likely to have comments, suggestions and new ideas. Some of them will write source code and add it to the existing source code in the CVS. Developers try and test the modifications, and may comment or even remove the newly added source code if they do not agree with it. Obviously, developers can disagree whether or not they think the source code is good enough. If they cannot reach an agreement, some of them can decide to take a version from the CVS before the change was made and add their changes to that version. This way two different versions are created. At a later stage, the community chooses which application should be taken as a starting point for further development.

The most important element is that the CVS leads to coordination since it allows a restricted amount of divergence. People who disagree with certain adaptations are challenged to prove that their suggestions are better - within the confinement of the community. The community then decides whose adaptation is best. Without this option of “in-house divergence” dissidents would be forced to prove their point outside the community. This could well be the beginning of a new competing version of the software. By allowing divergence, the CVS keeps the community together thereby easing convergence.

Another operational coordinating mechanism is the use of manuals. An example of a community that uses manuals is the Debian community. Debian has a number of different manuals, but they try to achieve the same goal, namely coordination. One maintainer in the Debian community explains:

---

23 E.g. jakarta.apache.org/site/agreement.html (May 21, 2003)
24 www.cvshome.org (August 2002)
“The policy manual enforces that kind of integration in Debian. The manual simply says, okay this is the way in which we put packages together. These are very simple rules saying that we will comply to the Linux file system hierarchy standard... Next to the policy manual there is a package manual that provides some technical details. For instance if you have binaries, which require one or more libraries then how do I make sure that the library is indeed installed on the computer? [It takes care of] dependencies between packages.”

One of the things the manuals prescribe is that each maintainer must classify the software. There are three classifications, namely ‘provides’, ‘depends’ and ‘conflicts. The classification provides should be used to explain what the software is supposed to do, e.g. Netscape provides Internet browser functionality. The classification depends is used to explain which other packages are needed to make the software work. The classification conflicts indicates that the software will not work properly if a certain package is already installed on the computer. A respondent: “In this system you assign relationships and that is what makes the system as a whole work.” In other words, it institutionalises coordination.

The Apache and the Linux community have no manuals, but have a coding style guide.25

**Coordination by authority**

There are several examples of coordination by authority in open source communities. Two main ones are addressed and illustrated here, that is, the coordinative effect of distributions (i.c. Debian) and the role of gatekeepers in OS communities (i.c. Linux and Apache).

Debian is an open source community that maintains the Debian distribution26. The distribution is based on the Linux kernel and many tools from the GNU project. It is the only significant Linux distribution that is not commercial.

Debian has an important converging effect, as one interviewee points out. “There are only a few users who choose between programs... Usually, the decisions are made by the distributions.” Debian, but also Red Hat and SuSE decide which applications and which version of the Linux kernel to include in their distribution. These three parties have a considerable market share, which lends authority to their distributions. If all three decide to include a certain application among competing ones, chances are high that the chosen one will become the most popular application.

There is quite some controversy over the role of Linus Torvalds in the Linux community. Many people argue that he clearly has leadership (e.g. Lerner and Tirole, 2002). Others argue the opposite. They view his role as a rather limited one, one with no formal authority over the Linux community (e.g. Van Wendel de Joode et al., 2003). Either way, his presence stimulates convergence. He maintains a version of the Linux kernel and is the only person with the prerogative to include additional pieces of source code in that particular version. In principle, people can ignore the decisions of Linus Torvalds and decide to build another version of the Linux kernel, but they don’t. Developers have faith in his decisions.28 This obviously creates a lot of convergence.

The structure of the Apache community is more ambiguous than that of the Linux community. Apache appears to work in a decentralized setting. Developers can directly contribute source code into the CVS (- if they signed the participation agreement). Bodies like the Apache Software Foundation (ASF) and Project Management Committees (PMC), however, do have the authority and power to impose decisions on the community. It is not clear whether they actually exercise their power, but they do have more authority and will usually be listened to more carefully than other random Apache developers. Therefore we consider them to be gatekeepers of the Apache community.

Comparing the two communities, Linux and Apache, only partly apply the same coordination mechanisms. See table 4. Table 4 shows, firstly, the differences in licenses. Linux licenses the software under the GPL, while Apache

---

25 The coding style guide indicates what the source code should look like and, for example, how comments should be included. See: www.apache.org/dev/styleguide.html (May 21, 2003) and www.linuxjournal.com/article.php?sid=5780 (May 21, 2003).
26 The Debian distribution comes with more than 8700 packages: www.debian.org/doc/manuals/project-history/ch-intro.htmlp.s1.1 (May 21, 2003)
27 Red Hat and SuSE are companies that develop, maintain and ship commercial distributions of the Linux operating system and a selection of applications.
28 According to an interviewee, Linus Torvalds decisions are more respected than decisions from others.
does so under its own license. The Apache license resembles the BSD license, and makes commercial exploitation of Apache software easier. However, for reasons discussed earlier, such licenses also create less incentive for convergence.

<table>
<thead>
<tr>
<th>Coordination mechanisms*</th>
<th>Apache</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Licenses</td>
<td>Apache Software License (BSD-like)</td>
<td>General Public License (GPL)</td>
</tr>
<tr>
<td>Trademarks</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Participation Agreement</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CVS</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Manuals</td>
<td>Coding style guide</td>
<td>Coding style guide</td>
</tr>
<tr>
<td>Gatekeeper</td>
<td>ASF &amp; PMC</td>
<td>Linus Torvalds</td>
</tr>
</tbody>
</table>

Table 4: Linux and Apache compared.

Secondly, Apache requires people to sign a participation agreement before they can contribute source code. The Linux community does not.

Thirdly, CVS is used in the Apache community and not – or only sporadically - in the Linux community. Currently, in the Linux community there is a lot of controversy about whether or not to adopt BitKeeper.30

Comparing both communities, we see that Apache applies more (institutionalised) internal coordination mechanisms than Linux does (e.g. CVS and the participation agreement). Such intra-community coordination increases overall convergence and compatibility. Lack of internal coordination needs to be compensated by external coordination mechanisms like standardisation. This would seem to explain the differences in coordination mechanisms between the two communities, among which the standardization efforts of the Linux Standards Base, efforts that are absent in the Apache community.

Conclusion

The tension between the incentive to change open source code (divergence) and the need for compatible open source software (convergence) can be addressed by different coordination mechanisms. We identified four categories of coordination mechanisms next to committee standardisation, i.e.: market or socio-economic coordination, regulatory coordination, operational coordination, and coordination by authority, and discussed their workings. These mechanisms complement standardisation, can partly be applied within the standards trajectory, and partly form an alternative to standardisation.

a. Complementary to standardisation:
Coordinative efforts other than standardization complement and support coordination by standardization. Stricter coordination within OS communities (internal coordination) appears to make committee standardisation, which is a form of external coordination, superfluous. Formulated differently, the more open the software development process, the more focus will need to be reintroduced at a later stage to achieve compatibility. Standardisation and certification function as the copingstone or ‘second chance’ for achieving compatibility. In other words, the more open the development process, the more likely standardisation will be required to recreate compatibility at a later stage.

b. Useful in standardisation:
Not all the mechanisms discussed in the previous are new to standardisation. Sometimes similar mechanisms are at work within standardization (e.g. formal and informal gatekeepers, rules on IPR, and psychological bandwagon effects). For example, the essence of CVS is comparable to the approach used in the IETF, where Internet standards procedures allow competing alternatives (i.e. in the expectation that the market will select the most useful one).

Further research is required to examine whether the standards setting could be helped by more explicit use of other coordinative mechanisms. One line of investigation would be whether it is possible to actively invoke the

29 The mechanisms reputation, level of activity and distribution have not been included in this table because they cannot be ascribed to one particular community. Reputation, for instance, is tied to individuals rather than communities. A distribution can decide to include Linux or Apache, but this is also not necessarily tied to a particular community. Finally, ‘level of activity’ is used by outsiders to compare communities (projects).

30 http://kt.zork.net/kernel-traffic/ (May 2003)
help of other coordination mechanisms in order to ease negotiations during standards development. That is, should standards bodies more actively encourage ‘pre-standardisation’? Should their policy aim for fortification of these ‘other’ coordination mechanisms?

c. Alternative to standardisation:
The mechanisms found in this research are also partly an alternative to standardization. This raises the question whether, in view of achieving compatibility, it matters which coordination mechanisms are used. For example, is standardisation an equally desirable coordination mechanism as for example the CVS? On what situational aspects would the answer to this question depend?

Finally, in this paper we focused on OS communities. These have many of the characteristics of network communities. As noted earlier, networks differ from markets (Van Wegberg & Berends, 2000). In networks all participants make a contribution, whereas in markets contributors and users are distinct groups. Can our insights be generalised to markets? Or does the difference between markets and networks mean that other coordination mechanisms are at stake? What are the theoretical implications? We recommend further research along this line of enquiry.

Bibliography
http://www.infonomics.nl/FLOSS/workshop/papers.htm