HANDLING VARIETY IN JAVA & LINUX:[1] COORDINATING MECHANISMS FOR THE FUTURE GENERATION?

RUBEN VAN WENDEL DE JOODE[2] & TINEKE M. EGYEDI
Faculty of Technology Policy and Management (TPM)
Delft, University of Technology

Abstract:
Development of the Future Generation promises to bring great variety in software, but also presents the danger of non-interoperability and market fragmentation if coordination mechanisms are not used effectively. This paper explores which mechanisms of coordination are helpful in limiting divergence in software development and deployment by examining two cases: Java and Linux.

A structural difference seems to exist between the mechanisms used by the two communities. For Java, divergence is where possible avoided ex-ante, whereas for Linux, divergence is foremost reduced ex-post. The conclusion discusses this difference and the implications of both types of coordination for interoperability.

The structure of this article is as follows. First, we introduce some theoretical notions that help identify mechanisms of coordination. We subsequently use these to examine the Java and Linux cases. The article closes with a comparison of how the two communities deal with divergence, and with a discussion about the wider implications of the research findings.

Introduction

In general, the source code of open source software (OSS) is widely accessible, freely available, and reusable. The most popular open source license, the General Public License (GPL), allows almost full use and reuse of source code.[3] Apart from engineering expertise, there are no significant barriers to join open source communities.[4] Low entry barriers also explain the variety among those actively involved in OSS development. One respondent we spoke to in the light of our research described the high level of variety in OSS projects: "(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."[5] More in particular, the individual hobbyist-developers – who still dominate OSS development[6]—are a source of variation because, as with other professionals, there is often a temptation to create new solutions rather than reuse solutions created by others.[7] Many developers in open source communities are reported to enjoy experimenting and taking risks, since their actions are not limited by deadlines or other fixed performance indicators.[8]

There is an increasing involvement of commercial organizations in open source communities.[9] Interviews[10] show that their interests and work setting can be very different from those of voluntary developers. Differences of a technical, organisational, political, and cultural nature are likely to result in conflicting software requirements and technical solutions. A board member of the Apache Software Foundation observes that, "a developer from, for example, IBM works more structurally towards deadlines and follows a certain approach; whereas a private developer, who does it as a hobby, is more creative, takes more risks by experimenting, and generally has a higher pay off risk ratio." Two other members of the Apache community note the increasing involvement of Sun people in Apache development. They note that Sun's deadlines are frustrating when working together with people who do it for fun.[11]

There is an unprecedented number of potential developers with different backgrounds and the possibilities to adapt source code, improve it, and thus create variety. On the one hand, these elements would seem to set ideal conditions for accelerated software evolution and innovation. However, on the other hand, variety may also cause grave compatibility and interoperability problems. For example, the diversity created by the localization and personalization of software to meet specific needs makes IT maintenance in large organizations a difficult and laborious task. Moreover, conflicting requirements may result in competing software branches ('forks").

The OSS setting offers the opportunity, the incentives, and the means to diverge. But does divergence occur to such a degree that it leads to interoperability problems and market fragmentation? In a debate on the subject only a few open source developers recognized the problem. Furthermore, some OSS packages have a large market share. For instance Apache has a 67% share in the webserver market[12] and Sendmail is the basis for most e-mail programs. This would not seem to be consonant with an unusual number of compatibility problems. All this suggests that open source communities somehow manage to limit divergent developments. In this paper we therefore focus on the following research question:

Given the high potential for divergence and the problems of interoperability to which this could lead, in what manner do open source communities limit divergence and increase convergence?

The aim is to identify a set of mechanisms that explains how two communities manage to avoid many of the interoperability problems which could be expected to arise. Standardization literature, in particular, is a suitable source of input to analyse interoperability issues. Here, we use the terms "market and technology coordination." We examine this body of literature to identify and discuss a number of coordination mechanisms. Notably, in this paper we limit the concept of coordination to interoperability. We do not address other aspects of coordination (e.g. how resources are gathered and mobilized, responsibilities and activities negotiated and structured, changes to the software made, and the quality of the software controlled).

The research is an explorative one. We study two communities: the Java and the Linux community. Both consist of a highly heterogeneous and geographically distributed group of programmers who have different and sometimes competing goals and interests. Both concern a platform technology, that is, a technology for which technical compatibility is crucial, and the economic stakes are high. This makes coordinated development more necessary and more difficult, respectively.

In contrast, both communities are organized very differently, about which more later. Together they promise to cover coordination mechanisms used in other open source communities. We base many of the observations on findings from a larger research program which includes OSS communities like Apache, Debian, and PostgreSQL.[13]

Theories of Coordination

Because standards are often equated with coordination, standardization literature is an important source of theory on coordination. Classic economic studies distinguish two types of coordination:[14] committee standardization and coordination by the market.[15] The distinction refers to standards developed by technical committees (committee standards) and to those that arise from a dominant market share (de facto standards).

We apply this categorization to organize and introduce a few theoretical notions below without aiming for a comprehensive overview. They are used here to trigger ideas about, and later on help identify mechanisms that, limit divergence in Java and Linux.

Committee Standardization

One of the most explicit coordination mechanisms is committee standardization. It is a response to technical and actor complexity, Schmidt and Werle[16] argue. In the technical sense, the need for coordination follows from the technical interrelatedness of components and products, that is, from the "network character of many technical artefacts."[17] 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 to 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 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."[18]

Market Coordination

Two concepts related to market coordination are of interest to OSS, : the "bandwagon mechanism" and "positive network externalities."[19] Farrell and Saloner[20] 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 coordination, that is, reduced diversity by technology selection, results as a by-product of this response.

There is also a socio-cognitive side to the bandwagon mechanism. Katz and Shapiro[21] mention *psychological bandwagon mechanisms.* But, we could include other group dynamic processes here too. For example, as the Linux case illustrates, in OSS communities indicators of software popularity induce bandwagon-like behaviours.

Katz and Shapiro identify three main sources of network externalities:[22]

- 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 of a certain kind sold, the more software developed for this type of computer)
- the quality and availability of post-purchase service for the product that depend on the experience and size of the service network. This may in turn vary with number of units of

the product 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).[23]

The notion of network externality provides a rationale for coordinative responses, and, sometimes, for the emergence of monopolies.[24] Though the notion is always discussed under the headings of "the market," the rationale of network externalities explains committee standardization 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, network externalities create a pull towards compatibility, as do other, market-bound rationales like "striving for economies of scale." It triggers coordination mechanisms. That is, although it is itself not a mechanism of coordination, it is relevant for OSS as an underlying rationale for coordination.

In sum, several ideas from standards literature seem relevant to coordination in OSS. In particular, we should note the common orientation provided by committee standards, the socio-cognitive and economic bandwagon mechanisms, and possible rationales for coordination such as network externalities and economies of scale. Their use below will allow us to identify and understand the mechanisms deployed in OSS to limit divergence.

The Case of Java

To understand the field of forces against which coordination of Java development took place, we briefly set the scene.[25] After the initial project phase, Sun started developing Java to achieve cross-platform compatibility. This represented a threat to companies with profits tied to platform-dependent products (e.g., Microsoft's Window applications). Because adaptation of the Java (computer- or browser) platform could undermine platform-independence and thereby the portability of Java programs, there was a temptation for companies with a high stake in a platform-dependent ICT-market to diverge from the specification and corrupt the platform. This would create incompatibility and market fragmentation and corrode network externalities. This sketches the background of Sun's attempts to safeguard Java's platform integrity. It explains why in the case of Java, coordination is often equivalent to control.

Although Java started out as a Sun project, Sun gave interested parties access to its source code. The company invited developers to comment on, experiment with, and improve the original source code. But, in the early years the decision to carry through changes in the original code remained Sun's.

> "One of the important things that we did when we first launched Java was to take the whole source of the system and put it out on our Web site. . . . There was the crowd of people who just play—they just had fun with it. . . . Then there were the people who were doing marginally useful things, like porting it . . . to lots of different platforms. But in the end, probably the most important reason for putting the source out was to allow people to scrutinize it. . . ."[26]

The instructive books which Sun authors and others wrote on Java, and the training programs that led to "certified" Java developers, contributed to a coordinated Java trajectory. These activities fortified the development of a Java practitioner community, but they did not guarantee interoperable Java implementations. The free distribution of the Java Software Development Kit (SDK, formerly JDK) proved more effective. The SDK contained a full suite of development tools to write, compile, debug, and run applications and applets. Use of the SDK narrowed down the number of possible programs to those that would run on "standard" Java platforms.

In the following, we discuss Sun's three most elaborate coordination instruments: licensing, Java Community Process, and committee standardization. They address different areas of coordination, which are: use and implementation of Java, the Java specification process, and the extended adoption of Java, respectively.

Licensing

The basis for Sun's licensing policy was intellectual property, which it protected by means of trademarks (e.g., Java-Compatible logo), patents (i.e., software algorithms) and copyrights (e.g., Java specifications). Integral to Sun's licensing policy were its test suites. These were used to certify compatible Java products, and were a prerequisite for using the "Java compatible" logo (the steaming cup of coffee). The logo had much goodwill among Java programmers, and worked as an incentive for making compatible commercial products.

Sun issued licenses with different degrees of restriction on Java use. Its company license for commercial use of Java most constrained the use of the technology. For example, Microsoft's license to Java, the *Technology License and Distribution Agreement* (TLDA, March 1996) stipulates that Microsoft must incorporate the latest updates of the Java platform in its products. Before Microsoft can use the Java-Compatible logo for marketing purposes, its products must pass the Sun test suites.

The Research Use license of the *Sun Community Source Licensing mo*del was the least restrictive. The new licensing model of December 1998[27] aimed to combine the advantages of the Open Source approach and the Proprietary licensing model. In it community members received broad access to the Java source code and were encouraged to partake in its development. In return, they were to share error corrections. To become a community member one had to sign a license agreement. Each Java developer could become a Research Use licensee free of charge with a simple click-through license. This license granted "broad experimentation and evaluation rights." With the Commercial Use license, users who wanted to deploy their Java products retained their intellectual property rights, but were required to open up their interface specifications. The model allowed them to use and modify Java source code. However, companies wanting to *sell* products based on modified source code would be charged a fee. Furthermore, products distributed under the Commercial Use license had to have Sun's compatibility logo and required a separate trademark license. Thus, Sun still kept control of essential developments.

Java Community Process

Licensing was in particular a means to coordinate—and in this case control—the use and implementation of Java. The Java Community Process (JCP) was a way to further coordinate development of the Java specification—i.e., due to the bandwagon mechanism Java had already selected as a de facto standard. With the new licensing model, Sun issued *The Java Community Process (sm) Program Manual: The formal procedures for using the Java Specification development process* (version 1.0, December 1998). In this manual, Sun unfolded

> ". . . a formal process for developing Java™ specifications . . . using an inclusive, consensus building process that not only delivers the specification,

> but also the reference implementation and its associated suite of compatibility tests."

The manual was a response to criticism that decisions regarding the specification of APIs were taking place behind closed doors. It discussed the procedures that were to be followed from the stage of requesting a new specification to its final release and maintenance.

However, there was heavy criticism of the procedures[28]. They excluded most independent Java developers, and Sun still vetoed decisions regarding eligibility of specification requests. The press spoke of a "gated community process."[29]

Sun's second version of the Java Community Process[30] answered much of the criticism. An Executive Committee was to represent "major stakeholders and is responsible for approving the passage of specifications through key points of the JCP . . . " It consisted of 16 members and a chair. Two of them were Sun employees: one, the committee chair, had no vote except to cast a tie-breaking vote; the other had a permanent voting seat on the Executive Committee. The other procedures were in important respects similar to those of many standards bodies: phased approach, voting procedures, expert groups, etc.

Formal Standardization

Sun announced early on that its aim was to have its de facto standard formally standardized by the Joint Technical Committee 1 (JTC1) of the International Standardization Organization (ISO)/ International Electrotechnical Commission (IEC). JTC1's approval would make customers, partners, and developers feel more confident about investing in it,[31] and the status of International Standard would make it easier for governments and industry standards bodies to refer to it.[32] That is, Sun aimed at a concerted type of bandwagon mechanism and a higher level of market coordination.

Sun took three initiatives to get JTC1 to formalize its Java specifications. See Figure 1. It started with a brief attempt to get its Java specifications accepted by the American National Standards Institute (ANSI), the US national member of JTC1. ANSI could then have proposed Java as a work item to JTC1. Soon after, Sun applied to become a submitter of a Public Available Specification (PAS) to JTC1. Subsequently, it turned to ECMA International, a standards consortium that has access to JTC1's Fast Track process. We will address the latter two most serious standardization attempts below.

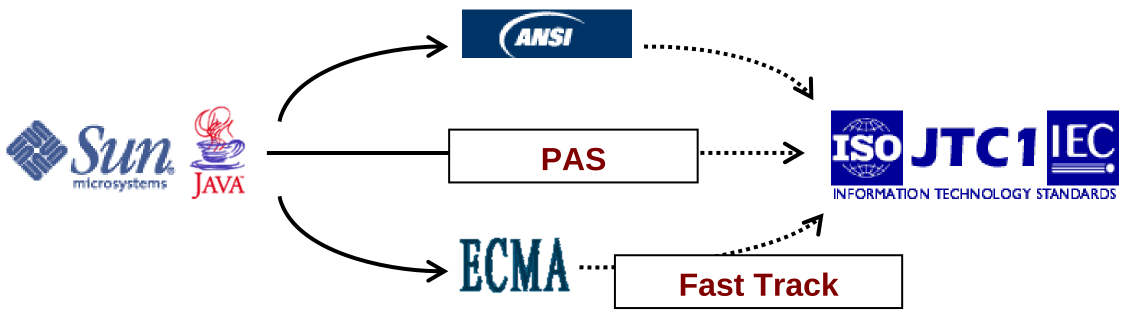

FIGURE 1: SUN'S THREE INITIATIVES TO STANDARDIZE JAVA IN JTC1[33]

JTC1 PAS Procedure

In March 1997, Sun applied to become an authorized Publicly Available Specification (PAS) submitter to JTC1 to get access to the PAS procedure. The procedure is a means for JTC1 to transpose specifications with high market relevance more rapidly into an international standard. Instead of passing through the usual stages of committee standardization, the specification starts out as a Draft International Standard (DIS), which, if approved by JTC1 members, immediately acquires the status of an International Standard (IS). Few hard conditions are made for organizations to be eligible as PAS submitters.[34] Although there is an expectation that the organization supports an open, multi-party and consensus driven approach, the criteria leave room for negotiation.

Sun, which positioned itself as the guardian of the Java Community Process, was the first commercial company to apply[35]. Sun's application was initially rejected.[36] The comments of the JTC1 national bodies indicated that there were hesitations about the way Sun would handle the evolution of the standard, IPR issues, and whether a Sun-led process would remain an open one. Sun responded to the comments in September 1997 and reapplied.[37] The company was accepted as a PAS submitter in November 1997. However, most votes in favor still included comments expressing concern about who was to be responsible for the maintenance and evolution of Java, about the influence of JTC1 national members therein, and about the openness of Sun's development process in competitive situations.[38] The national bodies expected Sun to address their comments in the Explanatory Report that was to accompany Sun's submission of the Java specifications.

Sun never submitted the Java specifications to JTC1. It doubted whether JTC1 would agree to ratify its work and leave the organization of a wider, participatory Java development process to Sun.[39] At that time, Sun did not want to surrender control of the evolution of Java.

ECMA

In April 1999, Sun approached the ECMA to discuss Java standardization. The ECMA, an international industry association for standardizing information and communication systems, is an A-liaison[40] member of JTC1. This status gives it access to the Fast Track process, similar to the PAS procedure. It allows a standard, regardless of its origin, to be put to vote as a Draft International Standard without going through the prior stages. In the past, ECMA standards had been submitted to a yes/no vote in JTC1 without modification.

During the negotiations about the program of work of ECMA TC41 on "Platform-Independent Computing Environments," Sun initially aimed for "passive maintenance" of the Java standard, which meant that Sun's Java Community Process would remain the primary arena for Java development.[41] Sun would then forward updates through ECMA to JTC1. However, the two parties ultimately appeared to agree on a program of work that left room for a more meaningful standards process in ECMA.

The first ECMA TC41 meeting took place in October 1999. During this meeting, Sun was to distribute the Java specification on CD-ROM. However, at the end of the meeting Sun announced that its lawyers required more time to consider the IPR issues involved.[42] In late 1999 it became clear that Sun would not contribute the Java specifications to ECMA TC41. The technical committee formally disbanded in March 2000. Sun withdrew officially because of a dispute with ECMA about the copyright of the specification. However, more probably, it feared that pursuing ECMA standardization could jeopardize its position in the lawsuit against Microsoft.[43]

In sum, the coordination mechanisms identified for Java, in particular regarding platform development, use, and implementation, served foremost to preserve the integrity of the platform and the interoperability of Java programs. However, there were two notable instances where Sun, despite its strategies of licensing, formalizing the Java Community Process, etc., could not forestall other players from actions that threatened to lead to fragmentation of the Java market. The first concerned Microsoft's way of using Java, for which Sun sued Microsoft. The second concerned real-time Java, the development of which partly took place outside Sun's realms.[44] These illustrate instances when Sun's attempt to coordinate and control Java development while safeguarding cross-platform compatibility failed.

The Linux Case

The Linux operating system started as a hobby project but grew to become one of the few serious competitors to Microsoft's hegemony on the desktop.[45] Although volunteers still dominate software development in Linux, the number of paid programmers is rapidly rising.[46] Linux developers have different backgrounds, different agendas and goals, and different software requirements.[47] In the following we discuss the coordination mechanisms that attempt to deal with these differences and that limit divergent software development. Information about these mechanisms has been gathered through a number of sources. The two most important sources of information are the secondary literature and sixty interviews conducted with developers from different open source communities. Most of them were either members of the Linux community or were otherwise related. Some of the interviews were in Dutch, so quotations from those have been translated into English.

Coding Style Guides

Linux has a *coding style guide*[48] that prescribes what the layout of a piece of source code should look like. Many Linux developers use the style guide. It prevents the following problem, noted by a Linux developer:

> "A Frenchman turned out to have changed the entire source code to the GNU style. Although, this is just a minor change, it does make it almost impossible to compare the programs because the indentation is completely different."

Another respondent explains in what manner such a guide may reduce software divergence. According to this Linux developer, a colleague developer had taken his application, made changes to the source code, and simultaneously changed the entire layout of the source code. The respondent, who also happened to be the maintainer of the source code, decided not to include the changes simply because it took him too much time to understand which changes had actually been made. The colleague developer then felt that his, in his eyes valuable, changes had unjustly been ignored and decided to start a new and competing project, thereby increasing divergence. In this particular case, if the colleague developer had used the coding style guide, the maintainer would have recognized and understood the proposed changes more quickly. The resulting process of arguing about which changes were (not) to be included would probably have given the other developer less incentive to start a new and competing project.

Respected Gatekeepers

One of the motives for developers to become actively involved in creating open source software is the desire to earn a reputation.[49] Reputation depends on the quality and the quantity of the contributed code. It has two coordinating effects. First, developers with a high reputation, for instance the project leaders of large communities, are generally better listened to than others. Especially, when a developer has little knowledge about a certain issue he or she will generally accept or agree with a decision made by a developer with a high reputation:

> "So then there is someone who has more knowledge about these things. You will automatically respect that person. You see his name everywhere and therefore he is likely to be right when he says something."[50]

The second coordinating effect is that a community of which one or more members is a developer with a high reputation will attract other developers.[51] The former project leader of Debian claims that he partly based his decision to get involved in a community on the presence of developers with high reputation: "Reputation does help: if you see a good name, then you will take a look."

Linus Torvalds, the founder of Linux, is an example of a highly respected developer in the Linux community. He maintains a version of the Linux operating system and he is the only person with the prerogative to include additional pieces of source code in that particular version. In principle, people can ignore his decisions and decide to build another version of the Linux kernel—and they do. However, developers who do not have the time to critically evaluate each decision will sooner adopt a decision made by him than by any other developer in the Linux community. Their decision is simply based on the fact that he has proven his programming skills and is therefore assumed to make better decisions than developers who have a shorter track record.[52] This obviously reduces the level of divergence in Linux.

Standardization Initiatives

Standards initiatives have taken place to increase convergence between the different Linux distributions and the different software applications these distributions comprise. First, although the market had its favourite, there was an effort 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 . . ."[53] The standard (LSB), first released in June 2001, "provides a way to ensure behavioral compatibility across Linux distributions and version releases. An application written to the standard will function the same across all LSB certified platforms . . ."

Second, to assure compliance with the standard, the Free Standards Group,[54] a non-profit group supported by industry that advocates standards for open source technologies, contracted with The Open Group, an organization that specialized in interoperability certification, to manage LSB Certification.

Third, inter-organizational coordination also improves compatibility in Linux. For example, the Free Software Foundation (FSF) frequently plays a mediating role among and negotiates with other open source communities, as the following quotation 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."[55]

Again, this is a good example of an initiative that reduces the level of divergence., The FSF has ensured that instead of having two competing versions developers will once again work on one version of the GCC.

Level of Project Activity

Developers want to download, install, and use the best-engineered application. They want to be sure it works and is reliable. However, they usually do not choose software based on the actual source code but on other indicators. The indicator most often mentioned is the level of maturity or activity of a project.[56] "Sourceforge and Freshmeat indicate a maturity level for each project, so you know that the software listed there is stable and works."[57] This level of maturity is a measure that combines the age of the project, i.e., community or development group, with the number of releases. Both are obviously not direct indicators of the quality of the source code itself. But, the argument is that if the community has been able to make many releases and is able to continuously attract new developers, then the software must be good.

Besides maturity, the websites also list statistics about the level of project activity. Freshmeat, for instance, maintains a so-called "popularity index."[58] SourceForge, which hosts a large number of open source projects,[59] uses several statistics to measure the activity in projects, such as the number of downloads. The activity level is frequently taken as an indicator of quality. It supports decision-making by individual developers, as several respondents confirm.[60] One of them explains:

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

Moreover, the more active the community is, the more likely it is that developers will download and use the applications, which further increases the activity level.

In sum, activity and maturity provide a very efficient and quick reference for estimating the quality of the software. The higher these levels are, the greater the chance the application has high quality and the less likely it is that users will create new and competing versions of the software. Therefore, indicators of project activity stimulate convergence.

Distributions

There are many OSS versions and variants that offer similar functionality. For most users and developers it is not an easy task to decide which ones to download and install. In most cases, however, they do not make the decisions: "There are only a few users who choose between programs… Usually, the decisions are made by the distributions."[61] A distribution is a collection of software programs issued as a package. Popular distributions are, for instance, Red Hat, SuSE, and Debian. The latter is the only one actually created in an open source community, while the other two are creations of commercial entities.

An OS community has an interest in getting its software included in a distribution: "It means a lot to us to be in such distributions, because we will get more exposure."[62] This exposure happens because many developers and users simply install the applications chosen by the manufacturer of the distribution.

Thus, because they select which applications and version of Linux to include, and because developers and users generally adopt this selection, distributions reduce divergence.

Comparing the Two Case Studies—In Conclusion

The Future Generation faces the danger of high levels of divergence, as illustrated by both the Java and the Linux community examples. This could result in compatibility and integrity problems. Yet, in both cases the level of divergence is much less than we would expect. A diverse set of mechanisms is responsible for limiting divergence and furthering coordination. The cases have two such coordination mechanisms in common, they are:

- *Instructive materials* (e.g., manuals and coding style guides), which explain how software should be written. These ensure that the software is written in a transparent, comparable way, which makes it easier to understand. As was argued, this is more likely to prevent the emergence of unnecessary variation and duplicate effort.
- *Standardization initiatives*. In both cases, initiatives are taken to standardize and formalize the software specifications. Standardization can play a major role in reducing divergence.

Some mechanisms, however, only seem to be present in one of the cases and not in the other. The difference can be understood in terms of *timing*: in the case of Java, most coordination mechanisms aim to prevent divergence from arising, i.e., ex-ante minimization. In the Linux case, however, the reduction of divergence is primarily addressed ex-post.

In the period discussed, Sun owned the intellectual property rights of the Java-core. Exceptions aside, this allowed the company to minimize the degree of divergence from the outset. Sun, for instance, asked prospective community members to sign a participation agreement to keep control of essential developments and prevent divergence. Another example is the Software Development Kit, the use of which leads to programs that run on a "standard" Java platform. The Sun compatibility logo, finally, is a mechanism that minimizes the chance of divergence by defining requirements that companies have to meet for them to use the logo in marketing campaigns. Essential here is that many methods used by Sun aim to minimize divergence ex-ante, that is, by keeping control of platform development.

The coordination mechanisms adopted in the Linux community are different. Consider for instance the activity level in a project. Knowledge of that does not directly affect the emergence of modified software. Instead, it stimulates convergence by influencing the choices of individual users and developers. It results in a choice for one or two alternatives on a collective level,[63] and affects the selection process in a way that may result in fewer alternatives (i.e., *passive convergence*). The same is true for Linux distributions and ex-post gatekeepers: they influence individual choices and stimulate convergence after divergence has occurred.
Table 1 lists the coordinative mechanisms identified and discussed in this paper.

Table 1—The Coordinative Mechanisms in Both Communities

		Java	**Linux**
Common	**Instructive material**	– Java manual, books, etc.	– Coding style guides
	Standardization initiatives	– Instalment of JCP – Formalization attempts	– Instalment of the LSB and FSG – Mediating role of the FSF
Different	**Ex-ante mechanisms**	– Sun compatibility logo – Software Development Kit – Participation Agreement – Licensing	
	Ex-post mechanisms		– Level of activity – Linux distributions – Gatekeepers

The question then is: what is the more effective and/or efficient model to reduce divergence, ex-ante or ex-post? [64] On first sight one might argue that ex-ante minimization of divergence is more efficient and effective, simply because it prevents divergence and thus creates less redundancy. It better safeguards the software's interoperability.

However, the answer to this question could be more complex. Research on public choice, for instance, suggests that involving stakeholders in a decision-making process will raise their acceptance of the end result.[65] Adopting this line of reasoning to software development could imply that, ultimately, allowing divergence to arise is more effective, because it reflects the involvement of the members of the community, which in turn promises better acceptance of the end result. Therefore, they would have less incentive to create different branches. Further research is needed to better understand the differences and consequences of ex-ante and ex-post reduction of divergence.

Some of the coordination mechanisms identified in this research are different from the mechanisms typically presented and discussed in standardization literature. Consider for instance the role of mechanisms like gatekeepers, participation agreements, and the activity level in a project. For standardizers it could be of interest to determine whether these mechanisms can be used. Are they complementary with existing mechanisms in standardization, can standardizers apply them, and/or do they form an alternative approach to standardization? The success of the Future Generation will need coordination. But the last word has not yet been said on which set of coordination mechanisms best suits a certain technology in certain setting.

Acknowledgements

We would like to thank the two anonymous reviewers of Computer Standards & Interfaces for their insightful and valuable comments on a previous version of this paper.

About the Authors

Ruben van Wendel de Joode is Assistant Professor at the Faculty of Technology, Policy and Management at Delft, University of Technology. He is part of the Dutch Institute of Government (NIG), the research school for public administration and political science. His research focuses on the organization of open source communities. He received two grants from the Netherlands Organization for Scientific Research (NWO) for research related to open source communities. The first grant was to study the interplay between intellectual property rights and open source communities. The results are published in *Governing the Virtual Commons* (Cambridge University Press, 2003). The second grant is for a research he is currently conducting on the reliability and continuity of so-called 'hybrid' open source communities. He has written numerous articles on open source, which have appeared in journals like *Electronic Markets*; *Knowledge, Technology and Policy*; *Computer Standards and Interfaces*; and the *International Journal of IT Standards & Standardisation Research*. He can be reached at <rubenw@tbm.tudelft.nl> or through his website at: < http://www.tbm.tudelft.nl/webstaf/rubenw/index.htm>.

Tineke M. Egyedi

Tineke M. Egyedi (PhD, 1996) is senior researcher in standardization at the Delft University of Technology, the Netherlands. Her present research and coordination tasks address issues of Interoperability (Sun Microsystems), Standards Dynamics (EU project, NO-REST), and Flexible Infrastructures (NGI/TU Delft). She is president of the European Academy for Standardization and associate editor and member of the editorial board of two international journals on standardization (CSI and JITSR, respectively). She has published widely (e.g., books, reports, book chapters, and articles).

Tineke has worked as a consultant and researcher in several companies and organizations. In these capacities, her activities included conducting studies for Dutch ministries (Infrastructure) and participating in several European projects (ELECTRA, SLIM, consortium standardization, SDOs and users). She was also organizer of EURAS2001 and the IEEE SIIT2003 conference.

---

[1] This paper is a revision of 'Handling Variety: The Tension Between Adaptability and Interoperability of Open Source Software', *Computer Standards & Interfaces*, 28 (1): 109-121.
[2] Corresponding author
[3] R. Stallman, *Free Software, Free Society: Selected Essays of Richard M. Stallman*, ed. J. Gay (Boston, MA: GNU Press, 2002).
[4] E. Franck and C. Jungwirth, Reconciling Investors and Donators—The Governance Structure of Open Source, Working Paper No. 8, Lehrstuhl für Unternehmensführung und -politik Universität Zürich

(2002), **http://opensource.mit.edu/papers/jungwirth.pdf**; S. Sharma, V. Sugumaran and B. Rajagopalan, "A Framework for Creating Hybrid-Open Source Software Communities," *Information Systems Journal* 12 (1) (2002): 7-25.
[5] M.L. Markus, B. Manville and C.E. Agres, "What Makes a Virtual Organization Work?," *Sloan Management Review* 42(1) (2000):15.
[6] G. Hertel, S. Niedner and S. Herrmann, "Motivation of Software Developers in Open Source Projects: an Internet-Based Survey of Contributors to the Linux Kernel," *Research Policy* 32 (7) (2003): 1159-1177.
[7] G. De Michelis, and others, Cooperative Information Systems: A Manifesto, in: *Cooperative Information Systems: Trends & Directions* eds. M. Papazoglou and G. Schlageter, 315-363 (San Diego, CA: Academic Press, 1997) 315-165; E.C. Valentin, A. Verbraceck and H.G. Sol, "Advantages and Disadvantages of Building Blocks in Simulation Studies: A Laboratory Experiment with Simulation Experts, in *Simulation and Industry;* 15th European Simulation Symposium, eds. A. Verbraeck and V. Hlupic (2003) 141-149.
[8] R. van Wendel de Joode, J.A. de Bruijn and M.J.G. van Eeten, *Protecting the Virtual Commons; Self-Organizing Open Source Communities and Innovative Intellectual Property Regimes* (The Hague: TMC Asser Press, 2003); R. van Wendel de Joode, "Variation and Selection in Open Source Communities," *Knowledge, Technology and Policy*, Special Edition on Evolution, 2004.
[9] Hertel, Neidner and Herrmann "Motivation of Software Developers,"
[10] More about these interviews later on.
[11] R. van Wendel de Joode, "Conflicts in Open Source Communities," *Electronic Markets* 14 (2) (2004): 104-113.
[12] www.netcraft.com (last visited July 23, 2004)
[13] The research program is partly funded by a grant from the Netherlands Organisation for Scientific Research (NWO: reference number 014-38-413); van Wendel de Joode, *Protecting the Virtual Common*s.
[14] J. Farrell, and G. Saloner, "Coordination Through Committees and Markets," *Rand Journal of Economics* 29 (2) (1988): 235-252; M. Katz and C. Shapiro, "Network Externalities, Competition and Compatibility," *American Economic Review* 75 (3) (1985): 424-440.
[15] Those who view committee standardization as a market process may feel somewhat uncomfortable with this categorization. Other objections to this categorization are possible as well (see the discussion on network externalities).
[16] S.K. Schmidt and R. Werle, "The Development of Compatibility Standards in Telecommunications: Conceptual Framework and Theoretical Perspective," in *New Technology at the Outset: Social Forces in the Shaping of Technological Innovations* ed. M. Dierkes and Ute Hoffman (Boulder, CO: Westview, 1992); S.K. Schmidt and R. Werle, *Co-ordinating Technology: Studies in the International Standardization of Telecommunications* (Cambridge, MA: MIT Press, 1998).
[17] Schmidt and Werle, *Development of Compatibility Standards*, 2.
[18] Schmidt and Werle, *Development of Compatibility Standards*, 6.
[19] 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).(Farrell and Saloner, "Coordination Through Committees and Markets.") 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).
[20] Farrell and Saloner, "Coordination Through Committees and Markets," 236.
[21] Katz and Shapiro, "Network Externalities."
[22] Katz and Shapiro, "Network Externalities."
[23] Product information is also more easily available for popular brands. Katz and Shapiro, "Network Externalities."
[24] For example, D. McGowan and M.A. Lemley, "Could Java Change Everything? The Competitive Propriety of a Proprietary Standard," *Antitrust Bulletin* 43, 715 (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.

[25] The period addressed covers the years up to 2001.
[26] Transcript of James Gosling's presentation at the JavaOne conference, May 29, 1996.
[27] R.P. Gabriel and W.N. Joy, Sun Community Source License Principles (SCSL), **http://www.sun.com/981208/scsl/principles.html**.
[28] E.g. by Rick Ross, founder of the JavaLobby and Wendy Fong of the Real-Time Java Working Group.
[29] E.R. Harold, The Java Gated Community Process, **http://metalab.unc.edu/javafaq/editorials/jcp.html**.
[30] Sun Microsystems, "The Java Community Process, Version 2, Review Draft of 2000.04.13," **http://www.jcp.org/aboutJava/communityprocess/review/jcp20/jcp2_draft_2000_04_13.pdf**.
[31] J.C. Perez, "Sun Changes Java Standards Strategy," Computerworld, , **http://www.computerworld.com/news/1999/story/0,11280,43097,00.html**.
[32] S. Shankland, "Sun Seeks Control of Java Process," CNET News.com, May 6, 1999, **http://news.com.com/2104-1001_3-225500.html**.
[33] T.M. Egyedi, "Strategies for *de facto* Compatibility: Standardization, Proprietary and Open Source Approaches to Java," *Knowledge, Technology, and Policy* 14 (2) (2001): 113-128.
[34] ISO/IEC, ISO/IEC Directives; *Procedures for the Technical Work of JTC1 on Information Technology*, 3rd ed., Supplement 1: "The Transposition of Publicly Available Specifications to International Standards," 1995.
[35] Other PAS submitters were at the time industry consortia such as ATM Forum and the Object Management Group.
[36] According to an interviewee, an anti- Java™ lobby supported by Microsoft complicated matters. Microsoft wanted its own Java functionality's enabled. Y. Clark, "Java Standard Voted Down—for Now," CNET News.com, July 30, 1997, **http://news.com.com/2100-1001-201952.html?legacy=cnet&st.ne.fd.mdh**.
[37] Sun Microsystems, "Sun response to ISO/IEC JTC1 N4811 and N4833," **http://www.Sun.com/**.
[38]ISO/IEC JTC1 N5090, November 1997.
[39] L. Bingley, "Sun Blames MS as Java ISO Plans Die, International Standards Organization Drops Request to Develop Java standard," IT Week, PC Week April 29, 1999, **www.zdnet.com/**.
[40] A-liaison membership is for organizations, which contribute actively in many JTC1 standards committees.
[41] Shankland, "Sun Seeks Control of Java Process."
[42] ECMA, Minutes of the 1st meeting of ECMA TC41 held in Menlo Park (USA) on 25th - 26th October 1999 (1999) ECMA/TC41/99/16.
[43] T.M. Egyedi, "Why Java^{TM} Was -Not- Standardized Twice," *Computer Standards & Interfaces* 23 (4) (2001): 253-265.
[44] T.M. Egyedi, "Strategies for *de facto* Compatibility: Standardization."
[45] P. Wayner, *Free For All: How Linux and the Free Software Movement Undercut the High-Tech Titans* (New York: Harper Business, 2000).
[46] Hertel, Niedner and Hemann, "Motivation of Software Developers."
[47] van Wendel de Joode, "Conflicts in Open Source Communities.".
[48] See: http://www.linuxjournal.com/article/5780 (May, 2003).
[49] Y. Benkler, "Coase's Penguin, or, Linux and the Nature of the Firm," *Yale Law Journal*, 112 (3) (2002), **http://www.yale.edu/yalelj/112/112-3ab1.html**; E. von Hippel, "Innovation by User Communities: Learning from Open-Source Software," *Sloan Management Review* 42 (4) (2001): 82-86; K. Lakhani and E. von Hippel, "How Open Source Software Works: 'Free' User-to-User Assistance," MIT Sloan School of Management Working Paper #4117, Boston, 2000, **http://opensource.mit.edu/papers/lakhanivonhippelusersupport.pdf**; Markus, Manville, and Agres, "What Makes a Virtual Organization Work?"; D. McGowan, "Legal Implications of Open-Source Software," *University of Illinois Review* 241 (1) (2001): 241-304; D. Selz, Value Webs, "Emerging Forms of Fluid and Flexible Organizations" (PhD. dissertation, University of St. Gallen, 1999).
[50] Based on an interview with a translator of KDE into Dutch.
[51] J. Ljungberg, "Open Source Movements as a Model for Organising." *European Journal of Information Systems* 9 (4) (2000): 208-216.
[52] According to an interviewee, Linus Torvalds decisions are more respected than decisions from others.
[53] www.linuxbase.org, May 15, 2003
[54] www.freestandards.org/about/press-fsg.php

[55] Cited in an interview with the vice-president of the FSF
[56] A project can for instance be a community, but could also be the group of developers that develops and maintains one specific module.
[57] Based on an interview with the coordinator of one of the New Hamsphire Linux user groups.
[58] The index is based on the number of subscriptions, URL hits and record hits. (software.freshmeat.net/stats/p.popularity, May 21, 2003)
[59] sourceforge.net/top/ (May 21, 2003).
[60] Cited from an interview with a developer from the Debian community, translated from Dutch.
[61] Cited from an interview with a Linux developer.
[62] Cited from an interview with the creator of LillyPond.
[63] van Wendel de Joode, "Variation and Selection in Open Source Communities."
[64] Since software development in open source communities is a cyclic process, the distinction between ex-ante and ex-post is intuitive, but analytic.
[65] I.S. Mayer, *Debating Technologies: A Methodological Contribution to the Design and Evaluation of Participatory Policy Analysis*, (Tilburg: Tilburg University Press, 1997).