J Manage Governance (2007) 11:165–177
DOI 10.1007/s10997-007-9022-9

ORIGINAL PAPER

Governance of open source software: state of the art

Paul B. de Laat

Published online: 9 June 2007

Abstract In this overview of governance mechanisms developed within open source software (OSS) circles, three types of governance are studied: 'spontaneous' governance, internal governance, and governance towards outside parties. Moreover, two main ways in which lessons from OSS can be applied elsewhere are explored: peer production of products other than software, and embedding 'peer-produced' products and peer processes into existing institutions ('coupling').

Keywords Commons · Configurations · Governance · Institutionalization · Open source software · Peer production

1 Introduction

The preceding contributions by Jørgensen, Hertel, O'Mahony, and Markus about governance of open source software (OSS) reveal an amazing repertoire of governance mechanisms. In this final overview I will weave several strands in their arguments together by distinguishing three types of OSS governance ('spontaneous' governance, internal governance, and governance towards outside parties). These roughly correspond to historical phases of both OSS development itself and studies *about* OSS. At the end, I venture some remarks about the 'lessons' of OSS governance to be applied elsewhere: peer production may apply to other kinds of products, and become embedded within existing institutions ('coupling').

P. B. de Laat (✉)
Faculty of Philosophy, University of Groningen, Oude Boteringestraat 52, 9712 GL Groningen, The Netherlands
e-mail: P.B.de.Laat@rug.nl

2 Point of departure: governance of the source code commons

OSS 'hackers' conceive of themselves as a movement to correct the failure of existing institutions (both industry and academia) to produce software adequately. By putting the source code of their programs on websites and constantly exchanging comments, patches, and new features, these hackers created a new institutional mode: peer production of software. As such, they exemplify a true 'community of practice', a globally distributed and virtual one at that.

As for the movement's ideology, two main currents can be distinguished. The more radical camp considers closed code software production as harmful, and espouses the view that software should be a public good, both publicly available and modifiable. To that end, they created a new regulatory framework, close to the public domain: the GPL that allows free use, modification, and (re)distribution of source code. However, one strict rule applies: any (re)distribution should carry the same license terms. Consequently, GPL-ed code will remain in the source code commons, in all its modifications and recombinations. These self-perpetuating regulations (sometimes referred to as 'viral') serve to protect the commons from private appropriation. To a large extent, the temptation to 'defect' is eliminated. As a corollary, the commons' resources remain up-to-date (thereby preventing the potential 'tragedy' of non-maintenance of the commons).

From the beginning of the movement a more moderate camp has also been in existence. To those hackers the main defect of industrial software production is its closed character, barring public inspection of the code. As a result, those programs are often inferior. OSS is considered the remedy to optimize software quality. To that end, a more relaxed kind of source code commons suffices, regulated by the BSD (and BSD-like) licenses. These allow almost anything to be done with the source code, apart from proper authorship notification. Their commons is (almost) unregulated and (almost) identical to the public domain. To them, protection from private appropriation is unnecessary.

These regulated commons are the point of departure for the whole OSS movement. The formal terms of open source licenses set the stage for a new institutional framework. It is important to stress this at the outset. As we shall see below, around the year 2000, hackers in larger projects renewed their attention to this defining framework, and decided to strengthen it by establishing non-profit foundational structures.

3 Phase one: 'spontaneous' governance

The first wave of OSS studies focused on the backgrounds and motivations of the volunteers involved. It soon turned out that the prevailing image of high school nerds fond of hacking had to be corrected. Although they are about 100% male indeed—as the stereotype would have it—, about half of them are in their thirties or forties, obtained academic ICT qualifications, have paid jobs, and are allowed to perform OSS work during work hours. As for the motivations involved (cf. Hertel, this issue), a mix of both internal and external motivation was found to be present.

Roughly in order of importance, volunteers enjoy the intellectual stimulus, want to learn and improve their skills, need the program involved for their work or their personal use, like to obtain recognition, or believe that source code should be open. A typical cluster analysis yielded four main types of motivation: intellectual stimulation and improvement of skills; a sense of obligation to the open source community; a need for the program for private uses; a need for the program for work (Lakhani and Wolf 2005).

Remarkably, these outcomes show that the movement is able to attract volunteers with a variety of motivations. In particular, *both* egoistic motivations ('rent-seekers') and altruistic motivations ('donators') are found to be present. This remarkable feat has been attributed to the GPL framework that governs most of the source code commons. It generates incentives for participation by *both* rentseekers and donors (Franck and Jungwirth 2003). Rent-seeking does not crowd out donations. As such, the framework may be considered one of the main conditions for the success of this social movement. Notice though that only a minority of hackers are staunch supporters of the cause of free software as a public good. But on the other hand, the large majority of OSS projects chooses to operate under GPL-conditions. This apparent paradox has as yet not been satisfactorily resolved.

The tacit assumption underlying these studies is that these communities of practice, crossing all institutional boundaries, are self-directing without any kind of explicit and formal coordination or control. Apart from the licensing framework, these communities are supposed to *spontaneously* create a stream of innovations. Any coordination and control that emerges, is the natural outcome of the differential number of contributions per person. Studies persistently show that a minority (about 20%) actually produces most of the code contributions (about 80%). This enables the high-performers to constitute themselves as *informal* leaders of the pack (cf. Muller 2006).

4 Phase two: internal governance

In the next wave of studies about OSS, this assumption was questioned. Spontaneous informal coordination turned out to be a relict from the past, at least for larger projects that had been in existence for a longer time. These projects employ a whole collection of explicit and formal tools for internal governance (i.e., coordination and control of OSS project work to achieve optimal outcomes; cf. Markus, this issue). The main tools can be usefully grouped into six categories: modularization, division of roles, delegation of decision-making, training and indoctrination, formalization, and autocracy/democracy.

(1) *Modularization*. In most projects, as the number of participants grew, the program was split in several modules. Literally dozens of modules may coexist together. This basic mechanism is almost universally employed across larger OSS projects. Similarly, the distinction between a stable and an experimental version is regularly employed in order to give both innovation and maintenance their own niches.

(2) *Division of roles*. Bundles of tasks are distinguished, and associated with differentiated access to project files. Several role divisions are in use, distinguishing from three up to eight roles. The common model as used on the Tigris platform, for example, reads as follows (cf. www.tigris.org, accessed February 2007). An *observer* is expected to participate in ongoing discussions on mailing lists; accordingly, (s)he has access to project documentation and files. A *developer* is expected to do likewise, but in addition to develop source code for patches and new features. For the purpose, (s)he can access the source code tree (but not officially change it). On top is the *project owner* who manages the whole project, both internally and towards the outside world.
Elsewhere, other role divisions can be found. The observer may be split into the 'user' (who only uses the software) and the 'contributor' (who also takes part in discussions). The developer may be split into the 'developer' proper (who may write code but not commit it to the official tree) and the 'committer' (who may). For large projects the project owner is usually seconded by several 'maintainers' (also known as 'module owners'). In due time, 'directors' may be appointed on top. A 'translator', a 'porter', a 'web designer', a 'technical writer', an 'international liaison', a 'super reviewer', a 'release manager' or 'driver'—the list of invented roles is endless. What matters here is, that role divisions may vary, from a small number of roles (mostly three) up to a much larger number.

(3) *Delegation of decision-making*. In OSS projects, decisions have to be made about the start of new activities, the methods to be used, the integration of modules into new releases, up to community-wide issues. One of the most important kind of decisions, however, has to do with code acceptance: which source code is accepted for incorporation into the experimental version, and—if it exists—ultimately into the stable product version? It is, of course, up to all participants (observers excepted) to actually create, test, and submit new source code. However, what level is entitled to make the decision about the subsequent actual inclusion of submitted code? The arrangements vary from centralized to decentralized designs.
In a centralized design, it is only the project leader (plus possibly some assistants) who is charged with such decisions; the classic example is the Linux kernel project with Linus Torvalds personally deciding upon all code changes to the experimental version. In a decentralized design, it is up to committers (or developers) in a given module to take these decisions. This can be found in projects such as Apache, Debian, Gnome, Mozilla, and Netbeans. After developers have tried and tested new code in their own personal copy of the source code tree, sometimes changes have to be reviewed first by fellow developers (once or even several times), *before* inclusion in the main tree. This might include taking a vote. In other projects and/or circumstances, committers may go ahead and import their changes in the official tree; reviewing via discussion boards (and/or voting) happens *after* code inclusion. The latter variety obviously speeds up the software development process

considerably (but at a risk). So in all, a wide range of designs exists, from centralized to decentralized designs.

(4) *Training and indoctrination.* In earlier times, conditions for entry were none. However, while projects grew larger and a division of roles imposed itself, entry requirements came to be formulated. At Debian, e.g., contributors who desire to become developers (with commit privileges) have to succeed a three-step application process with a Debian developer ('sponsor') (cf. Coleman and Hill 2005). They have to prove their identity (by having their cryptographic key signed face-to-face), prove their knowledge of and allegiance to OSS philosophy, and demonstrate their technical competences. Similar 'examinations' exist inside freeBSD and Mozilla, focusing upon technical competences only (Holck and Jørgensen 2003/4).

(5) *Formalization.* Several formal tools and procedures have been invented in the OSS arena in order to knit globally distributed virtual contributors together. Mailing lists and newsgroup archives enable discussion to take place. Tools like Bugzilla standardize bug reporting and the raising of issues for discussion. Versioning systems (from CVS and Subversion up to GIT) let distributed authors work together simultaneously on the code tree and keep track of all changes. For testing purposes tools such as tinderboxes and 'verification machines' compile source code on a regular basis (cf. Jørgensen, this issue). Elaborate procedures have also been developed, in particular for testing, reporting bugs, and sending in new code. As regards all these tools and procedures, I am not aware of any serious objections raised against them. Moreover, the whole OSS community seems to have effectively standardized on the use of them, a remarkable achievement (Robbins 2005). Therefore, taking up a distinction coined by Adler and Borys a decade ago (1996), I interpret this formalization as 'enabling' (allowing hackers to better perform their tasks), not as 'coercive' (an attempt to force compliance).

(6) *Autocracy/democracy*. Is leadership self-appointed and imposed from above (autocracy), or are community members empowered to choose their leaders in some kind of electoral process (representative democracy)? Autocracy is evident in Linux; Linus Torvalds initiated the project and has remained the leader ever since (at least for the kernel). In many smaller OSS projects their founders also stay on as autocratic leaders. In a similar vein, self-electing committees at the top may hold power (like in Perl and Mozilla). On the other hand, democratic processes for electing leadership may be instituted. Examples involve Debian, where the Project Leader is elected annually by developers; freeBSD, where committers choose the 'core team'; and Netbeans, in which developers choose the governing board. Project leaders function as *primi inter pares*.

On the basis of this array of governance tools, OSS communities assume some overall design. Within the multitude of possible OSS designs, I will now venture to

identify typical OSS *configurations*, i.e., combinations of internally consistent scores on OSS governance dimensions. As Markus (this issue) forcefully argues, the search for configurations is to be considered the way forward for OSS analysis. Just as Henry Mintzberg distinguished essential configurations for hierarchical firms in 1979 (Mintzberg 1979), this needs to be done for volunteer OSS communities now.

Notice that for all OSS designs, some kind of hierarchy is in evidence. While tasks may be chosen voluntarily, access to files has to be granted, new code for inclusion be approved of, code freezes be respected, tools and procedures be accepted. This is, however, not the usual hierarchy, but one of *esteem*. If volunteers want to obtain esteem from the project leader(s) of a particular project, they have no choice but to play by their rules, or leave. In view of this hierarchy nexus, there is a close analogy with the usual non-volunteer organizations. To put this correspondence into relief in my configurational analysis, each of the OSS governance dimensions as distinguished above will be 'paired' with its more usual organizational analogue: (1) 'modularization' corresponds to 'horizontal differentiation', (2) 'division of roles' to 'vertical differentiation', (3) 'delegation of decision-making' to 'vertical decentralization', while the three remaining OSS dimensions have the same denominations in organizational analysis. This correspondence will allow comparisons of OSS configurations with the usual hierarchical configurations (of which Mintzberg's Pentagon is a prime example). For ease of exposition, I will henceforth mainly use the 'usual' organizational dimensions—but it has to be borne in mind that they may have a specific, slightly different meaning in the context of OSS.

With this analogy in mind, my conjecture about OSS configurations is, that these tend towards—at least—two 'natural' configurations. On the one hand, communities may originate as autocratic ventures. Being small, one leader provides enough structure. When they grow larger, coordination and control of modules (horizontal differentiation) require ever more attention. Given autocratic control, it is most consistent for leaders to move towards a high degree of vertical differentiation, vertical centralization, and formalization. Employing the distinction between 'mechanistic' and 'organic' systems of management as coined by Burns and Stalker decades ago (1961), I refer to this configuration as an 'autocratic-mechanistic' structure (Table 1). On the other hand, from the outset communities may be run as a cooperative with collegial control (direct democracy). Upon expansion and differentiation into several modules, these communities, in keeping with their egalitarian ethos, will gravitate towards representative democratic procedure, as well as keeping the amount of vertical differentiation and vertical centralization as low as possible. Formalization, as the means *par excellence* to connect virtual members together, remains indispensable. At the same time, training and indoctrination assume paramount importance. A 'democratic-organic' structure ensues (Table 1).

Due to the correspondences established above, a comparison with Mintberg's configurations can easily be made. The 'mechanistic' part of the first configuration is close to Mintzberg's 'machine bureaucracy', though in keeping with the network character of OSS projects the bureaucracy involved is more properly interpreted as

Table 1 Two ideal type OSS configurations and their design parameters

Design parameters	Autocratic-mechanistic structure	Democratic-organic structure
Autocracy/Democracy	Autocracy	Democracy
Horizontal differentiation (modularization)	High	**High**
Vertical differentiation (division of roles)	High	Low
Vertical centralization (delegation of decision-making; in reversed order)	High	Low
Formalization	High	**High**
Training and indoctrination	Low	High

Remarkable scores in bold type (see text)

an 'enabling bureaucracy' à la Adler and Borys (1996). The 'organic' part of the second configuration is close to Mintzberg's 'professional bureaucracy', though remarkably enough with a high degree of formalization (while 'enabling'). So contrary to the usual typifications, a high degree of horizontal differentiation (in modules) and formalization (in the 'enabling' sense) are a hallmark of *both* OSS configurations (cf. bold type in Table 1). In order to attract and retain virtual volunteers in great numbers from all over the globe, these features are (to be) retained at all times.

Of course this is just an *idealtypische* kind of exercise; actual configurations of OSS communities will show a lot more variation. Moreover, the exercise refers to *larger* communities only, with several hundreds of members. As long as projects remain confined to some dozens of people—which applies to the majority of projects on open source platforms—, a 'simple structure' suffices (a strong project leader that personally coordinates developers and observers). Future research will have to connect these two ideal type configurations to relevant contingencies, such as the age of the project, the size of the community, and the type of software involved (see Ye et al. 2005, for one of the first exercises of the kind).

This distinction between two configurations also enables some answers to the question posed in the call for papers that started the JMG roundtable about OSS governance (cf. de Laat, Introduction (...), this issue): does governance rely on control, trust, or both? It would seem that whenever projects gravitate towards a democratic-organic regime, trust is the main mechanism, trust in the *double* sense of trusting project members to work loyally towards the project's goals and trusting them to sensibly choose its leadership. An autocratic-mechanistic regime expresses quite the opposite: participants as both workers and voters are highly distrusted, control (in the coercive sense) of their behaviour on both counts is mainly relied upon. High formalization figures in *both* regimes, while this is a control that is enabling, not coercive out of distrust.

At the present day, worries are mounting about hackers' contributions of source code. Code may be sloppy, have bugs, contain Trojan horses (viruses), carry a non-compatible license, or introduce patented matter. Reverting such changes from the main tree of source code is a nuisance. Organic arrangements are especially sensitive to these dangers. How are such projects likely to react? As a first measure

of defense, future participants will be scrutinized: screening their identity, assessing their loyalties and technical competences (cf. Debian). So distrust is dealt with at the gates of *entry*, not inside. If these measures are not enough, it is to be feared that the level of trust as granted by design parameters will be reduced and control over contributors tightened (more vertical centralization and vertical differentiation). Distrust is dealt with *inside*, and contributes to shifting the regime towards the mechanistic end.

5 Phase three: governance towards outside parties

A third type of OSS studies called attention to tendencies towards increasing institutionalization. When projects grew larger and achieved more success, hackers could no longer ignore the world outside. Firms, national and international organizations, as well as non-governmental organizations are fascinated by OSS, and want to jump on the bandwagon. At the same time, the institutional space as defined at the outset of the movement, the regulated source code commons, is in danger. The main threat is the increase of software patenting (in the US alone, about 20,000 patents are granted per year). Therefore, OSS developers are increasingly likely to infringe upon patents held by other parties (or issuing soon). Lawsuits are ever more likely. Ultimately, this could bring the commons to a collapse, and with it, the foundations of the experiment of OSS as a whole.

In order to deal with these challenges and threats, projects were forced to consciously manage their relations with parties outside of OSS. The dominant solution is to create a legal shell around the OSS undertaking, that furthers and protects the project's interests. This also creates an official spokesperson for a project. Usually, a non-profit foundation is established. Starting with Debian in 1997, accelerating from 1999 onwards, up to Linux in 2007, all major open source projects have incorporated (cf. O'Mahony 2005). These foundations typically handle donations, both money and hardware. Moreover, they uphold copyright licenses (the GPL in particular), trademarks, and brand names. For the purpose, actual code contributions may have to be formally licensed to the foundation, or even the copyrights be handed over (cf. O'Mahony 2003). Similarly, the foundations take care of any charges of patent infringement against their members. Often, project and foundation have about the same membership: active community members have the right to become members of the foundation, and as such may elect its Board of Directors (cf. yearly elections at Debian, Apache). Project and foundation membership may also be distinct. Take for example the Linux Foundation: membership is reserved for external partners like firms and universities, which have to pay a high entry fee.

Notice, that as soon as an OSS community of the democratic-organic type (cf. Table 1) has created such a foundation as a protective shell around it, the prototype of the 'community managed model of governance' has evolved, as so eloquently described by O'Mahony in this issue. Next to pluralism, representation, decentralized decision-making, and autonomous participation, the essential value of independence is finally ensured as well.

The further development of relations between foundations and actual projects promises to be a fascinating area of study. As a rule, foundations are set apart from the actual project, in order to guarantee non-interference with its proceedings. However, due to external pressures, foundations may gradually come to *dominate* the actual project. At Apache, for example, the elected Board of Directors of the Apache Software Foundation does have a say in project work (by appointing the Project Management Committees that oversee the modular projects; see http://apache.org/foundation/how-it-works.html, accessed February 2007). At Gnome, to take another example, its foundation has managed to acquire a say in planning release dates and contents of product versions (in order to accommodate industrial preferences) (O'Mahony 2005). These are forebodings of foundation and project becoming ever more intertwined. If a democratic-organic regime has evolved within communities, a sizeable transformation of structure might be the outcome. In accordance with the 'Iron Law of Oligarchy' (as formulated by Michels) the regime might well gravitate towards a less democratic and less organic form—if not de jure, at least de facto. Ultimately, larger OSS projects are in danger of ending up being governed like a mirror image of hierarchical firms. Democratic and organic rule (cf. Table 1) would become a relict of the past.

6 Lessons from OSS

Towards the end, the call for papers for this roundtable about OSS governance addressed the question: to what extent governance mechanisms for OSS may have applications elsewhere (cf. de Laat, Introduction (...), this issue)? It would seem that the 'lessons' of OSS peer production can be generalized in two ways: towards other types of products, and towards existing institutional contexts (cf. O'Mahony, this issue, about diffusion and adaptation of the OSS model).

6.1 Type of product

Has 'commons-based peer production' (Benkler) taken root elsewhere? The institutional innovation of open source licensing (especially the GPL) has inspired several other 'content producers' in the sciences, technology, and art to reveal their creations to the general public as well. It was up to the non-profit Creative Commons corporation (Lessig) to create a systematic licensing framework which is now used all over the world. Their licenses have created a variety of public-domain-like commons for intellectual resources, and actually offer more options than defined by open source licenses in particular (cf. de Laat 2006). This is, however, mostly a movement for open *access* only, not for peer *production* as well.

Some notable exceptions to this rule exist. For one thing, an initiative called 'open source biology' is being contemplated, which at the moment focuses in particular upon virtual genetic engineering by means of software (Hessel 2006). For another, amateur content producers have developed quite remarkable initiatives. Next to computer users making their idle computer time available for solving radio astronomy problems (SETI@home project), and amateur astronomers marking

craters on Mars (NASA Clickworkers) as well as exploring outer space (as recounted in Shah 2006b), the most prominent endeavour is the Wikipedia community, aiming at the creation of a free on-line encyclopedia and other reference works (Table 2). Their governance is almost an exact copy of a GPL-like OSS community. Created entities are licensed along GPL-like terms (for text), while internal governance, though autocratic as yet, is very organic (with training and indoctrination still low).

However, voluntary innovation is not only to be observed for 'informational' products, but also for *physical* products. As explored by Eric von Hippel and coworkers like Nikolaus Franke and Sonali Shah in a series of studies (Franke and Shah 2003; von Hippel 2005), sports equipment (for windsurfing, mountain biking, and snowboarding) is invented and tested by enthusiastic communities of sports fanatics. These 'lead users' assist each other for free, and 'freely reveal' innovations to one another. For this phenomenon, von Hippel coined the umbrella term 'user innovation' (which is meant to cover not only physical products, but also the informational products as discussed above) (Table 2).

What about the governance of 'user innovation'? It will be a future challenge to compare user networks for physical products with those for software (or informational products in general). At first glance, the former differ from the latter. The user-innovator sportsmen involved did not bother to patent their innovations or impose trade secrecy, so their new designs became a public good, unprotected from private appropriation. Moreover, as far I can judge from the description of these networks by von Hippel *et alii*, no internal governance—let alone towards outside parties—as defined above seems to develop. Such networks for physical product innovation may be considered communities of practice *without* formal governance. The causes and implications of these large differences in governance remain to be explored. Relating back to the historical phases of OSS governance described above, my conjecture would be that

Table 2 Institutional embeddedness of 'freely revealing' user innovation communities for informational and physical products

Type of product		Communities outside institutions ('peer production')	Communities and institutions loosely coupled (incorporating the product)	Communities and institutions more tightly coupled (incorporating peer production)
Informational products	Software	OSS	Open source companies	• Corporate open source networks • Developer networks • Enterprise networks
	Reference works	Wikipedia	–	–
Physical products	Sports equipment	User innovation networks	Manufacturing user-generated innovation	Providing users with platform products and design tools

Sources: de Laat (2004), von Hippel (2005)

communities for physical product innovation, while not globally distributed but (mostly) locally concentrated, are unlikely to evolve towards 'maturity' in the same way as OSS by developing internal governance and governance towards external parties. They are more likely to remain stuck, as it were, in phase one of 'spontaneous' governance.

6.2 Institutional embedding

On the other hand, the phenomenon of OSS—or, rather, user innovation networks generally—has become a source of inspiration for established institutional actors: how to learn from the dynamics of such 'open innovation', and bring its products or its dynamics *within* their own sphere? Let me first discuss the case of OSS proper. Although some university departments reportedly have adopted the OSS model for the diffusion of their internally produced software (on GPL terms), OSS as a model is mainly taken up by companies. As a first step, OSS as a *product* is being 'free-rided' upon: firms tie services to the free product (like customization, consulting, training of personnel), prepare their own hardware for it, or sell closed commercial applications on top of it ('open source companies'; Table 2). Notice though, that many companies do not only take a free ride, but also voluntarily donate back internally developed source code to the OSS project involved (cf. Henkel 2006). All the while, communities and firms remain (largely) separate or 'loosely coupled'.

The next step involves taking OSS seriously as a *peer production* process. It is an effort to emulate *work* conditions as typical for OSS *within* the industrial context (cf. Hertel, this issue). The main kinds of such 'tighter coupling' are the following (Table 2; cf. de Laat 2004):

(1) *Corporate open source networks.* Software corporations decide to open up specific technologies to hackers in general, upon OSS licensing terms, by mounting a production platform on the Internet. Examples that come to mind are Mozilla and Netbeans;
(2) *Developer networks.* Another coupling involves opening up a hard- or software platform, whereupon outside developers are invited to write software applications in order to enrich its value. An OSS community is created but only for those that identify themselves as interested developers and subscribe to licensing terms that keep the source code within the confines of the community. Examples involve Motorola with its iDEN handsets network, and Nokia with its Open Standards Terminal software platform;
(3) *Enterprise networks.* As a final possibility, open source networks for software development may be installed *inside* corporations, behind the corporate firewall, connecting the firm's globally distributed subsidiaries. Companies like HP and SUN employ such practices.

Along these forms of tighter embeddedness, the main parameter that changes is *access*: smaller circles of admitted members are delineated. Associated with this, licensing becomes more restrictive, sharing being permitted only among insiders. Communities effectively become 'gated'.

How are gated communities being governed? Which mechanisms of proper OSS governance are copied, if any? As for 'corporate open source networks', relations with the founder firm are an important variable. If, in due time, the network is spun off as an independent venture (this happened to Mozilla and Eclipse), it will evolve as a 'proper' OSS community (cf. the discussion in O'Mahony, this issue, about sponsor founded versus community founded projects). However, the founder firm may prefer to retain influence. Concerning these production platforms, still under sponsor control, and the very similar 'developer networks', empirical data about governance are lacking as yet. I would argue, though, that a whole array of internal governance tools is likely to be employed just as for OSS proper (cf. phase two above). In particular, if a company controls incorporation of source code and formally owns the code tree (cf. the 'gated source' community analyzed in Shah 2006a), an autocratic-mechanistic structure is to be expected. In 'enterprise networks', ultimately, governance is determined by company rules. Licensing proceeds along the closed code model, and volunteers ultimately have to obey the company hierarchy. The only thing remaining from OSS governance might be the formidable collection of software development tools for virtual collaboration.

What about user innovation in *physical* products as a source of inspiration for established companies? Coupling mechanisms, as described by von Hippel (2005), are quite similar to those just described for OSS. Firms do occasionally incorporate innovative designs from lead users and proceed to mass manufacture them ('free-riding'); or the lead users involved themselves establish a company precisely for the purpose ('lifestyle firms'). Notice that this is the only way in which these innovative designs can be mass produced and distributed. The investments involved are simply too high for user innovators on their own (as opposed to the case of informational products, the Internet allowing instant distribution at no cost).

As a final step, producers of physical products may try, just like producers of software, to introduce peer production within the company (tight coupling). Experiments as mentioned by von Hippel (2005, ch. 9) include the provision of platform products and toolkits for design. Hopefully, these will help users start innovating. Recently, the LEGO Group has even provided software design tools for its young clientele. Notice that these initiatives are quite analogous to the corporate open source networks and developer networks for OSS previously described (Table 2). As for the study of governance mechanisms that apply to embedded networks for physical products, it is quite virgin territory—research data are lacking. Sporadic evidence suggests, that such networks are not interconnected, but consist of empowered individuals tied together by a focal point (hub-and-spoke structure). Remarkably, 'enterprise networks' for physical products, although conceivable, are mentioned nowhere.

References

Adler, P. S., & Borys, B. (1996). Two types of bureaucracy: Enabling and coercive. *Administrative Science Quarterly, 41*(1), 61–89.

Burns, T., & Stalker, G. M. (1961). *The management of innovation*. London: Tavistock.

Coleman, E. G., & Hill, B. (2005). The social production of ethics in Debian & free software communities: Anthropological lessons for vocational ethics. In Koch (2005), pp. 273–295.
de Laat, P. B. (2004). Evolution of open source networks in industry. *The Information Society, 20*(4), 291–299.
de Laat, P. B. (2006). Internet-based commons of intellectual resources: An exploration of their variety. In J. Berleur, M. I. Nurminen, & J. Impaglizzo (Eds.), *Social informatics: An information society for all?* (pp. 171–183). IFIP International Federation for Information Processing, Vol. 223. Boston: Springer.
DiBona C., Cooper D., & Stone M. (Eds.) (2006). *Open sources 2.0: The continuing evolution*. Sebastopol: O'Reilly.
Feller J., Fitzgerald B., Hissam S. A., & Lakhani K. R. (Eds.) (2005). *Perspectives on free and open source software*. Cambridge, MA: The MIT Press.
Franck, E., & Jungwirth, C. (2003). Reconciling rent-seekers and donators - The governance structure of open source. *Journal of Management and Governance, 7*(4), 401–421.
Franke, N., & Shah, S. (2003). How communities support innovative activities: An exploration of assistance and sharing among end-users. *Research Policy, 32*, 157–178.
Henkel, J. (2006). Selective revealing in open innovation processes: The case of embedded Linux. *Research Policy, 35*, 953–969.
Hessel, A. (2006). Open source biology. In DiBona et al. (2006), pp. 281–296.
Holck, J., & Jørgensen, N. (2003/4). Continuous integration and quality assurance: A case study of two open source projects. *Australasian Journal of Information Systems, Special Issue*, 40–53.
Koch S. (Ed.) (2005). *Free/open source software development*. Hershey: Idea Group.
Lakhani, K. R., & Wolf, R. G. (2005). Why hackers do what they do: Understanding motivation and effort in free/open source projects. In Feller et al. (2005), pp. 3–21.
Mintzberg, H. (1979). *The structuring of organizations*. Englewood Cliffs, NJ: Prentice Hall.
Muller, P. (2006). Reputation, trust and the dynamics of leadership in communities of practice. *Journal of Management and Governance, 10*(4), 381–400.
O'Mahony, S. (2003). Guarding the commons: How community managed software projects protect their work. *Research Policy, 32*, 1179–1198.
O'Mahony, S. (2005). Nonprofit foundations and their role in community-firm software collaboration. In Feller et al. (2005), pp. 393–413.
Robbins, J. (2005). Adopting open source software engineering (OSSE) practices by adopting OSSE Tools. In Feller et al. (2005), pp. 245–264.
Shah, S. (2006a). Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science, 52*(7), 1000–1014.
Shah, S. (2006b). Open beyond software. In DiBona et al. (2006), pp. 339–360.
von Hippel, E. (2005). *Democratizing innovation*. Cambridge, MA: The MIT Press.
Ye, Y., Nakakoji, K., Yamamoto, Y., & Kishida, K. (2005). The co-evolution of systems and communities in free and open source software development. In Koch (2005), pp. 59–82.

Author Biography

Paul B. de Laat is Assistant Professor of Philosophy of Science, Technology & Society, at the Faculty of Philosophy, University of Groningen (Groningen, The Netherlands). He obtained a Masters Degree in theoretical physics (University of Utrecht) and a PhD in organizational sociology (University of Amsterdam). His current research focuses upon governance of open source software, diffusion of open source networks, commons for informational resources and the associated IPRs, and the role of virtual trust in cyberspace.