Coordination Processes in Open Source Software Development: The Linux Case Study

Abstract

Although open source projects have been subject to extensive study, their coordination processes are still poorly understood. Drawing on organization theory, this paper sets out to remedy this imbalance by showing that large-scale open source projects exhibit three main coordination mechanisms, namely standardization, loose coupling and partisan mutual adjustment. Implications in terms of electronically-mediated communications and networked interdependencies are discussed in the final sections where a new light is cast on the concept of structuring as a by-product of localized adjustments.

Key words: coordination, interdependence, standardization, loose coupling, structuring.

By Federico Iannacci,

Department of Information Systems,

LSE, Houghton Street, London WC2A 2AE

April 2005; Please send comments to: F.Iannacci@lse.ac.uk

1 Introduction

The issue of coordination in open source software in general and Linux in particular has poorly been explored by scholars and practitioners alike. For instance, in a recent paper presented at the FLOSS workshop, Crowston (2002) has maintained that "[i]n an OSS project, a range of coordination mechanisms seem to be used, but exactly what these practices are has not been studied in detail". By the same token, Weber (2004: 11-12) has insightfully asked at the beginning of his book: "[h]ow and why do these individuals coordinate their contributions on a single focal point? The political economy of any production process depends on pulling together individual efforts in a way that they add up to a functioning product. Authority within a firm and the price mechanism across firms are standard means to efficiently coordinate specialized knowledge in a complex division of labor– but neither is operative in open source".

Instead of studying the coordination processes characterizing all types of open source projects, this paper focuses on one of such projects, namely Linux, because coordination is remarkably more interesting and challenging in large-scale projects (Lanzara and Morner 2003: 10) considering that size brings about a host of issues such as the problem of information overload bearing on the project leader(s), the increased possibility of conflict let alone the need to motivate a growing number of developers to keep momentum.

The remainder of this paper unfolds in the following fashion: section two overviews the literature on open source software development by identifying two separate threads of thought; section three highlights the methodology I have used to address the problem of coordination; section four spells out the coordination mechanisms characterizing the Linux kernel development while section five outlines the major contributions deriving from my argument.

2 Literature review

Although there is a burgeoning literature on open source software development, one can identify two main threads of thought, namely the social and the engineering stream (Feller and Fitzgerald 2000)[1]. While the social stream analyzes the social dynamics of open source software development by drawing on such ideas as gift economy (Raymond 1999), social structure (Healy and Schussman 2003), user-based innovation (von-Hippel 2001), complexity/chaos theories (Nakakoji, Yamamoto et al. 2002; Muffatto and Faldani 2003) and

[1] For the purposes of the present discussion, the business models thread is not being investigated because it focuses on the way corporations have used the open source paradigm to make a profit which, obviously, falls beyond the problem of coordination I am trying to address.

transaction costs (Benkler 2001), the engineering approach attempts to situate the open source model within the context of other systems development methodologies (Feller and Fitzgerald 2000).

Despite this enormous research effort, it seems that the issue of coordination has not yet been fully addressed to the point that scholars contend that "little is known about how people in these communities coordinate software development across different settings, or about what software processes, work practices, and organizational contexts are necessary to their success" (Crowston and Howison 2005).

But what is coordination? Broadly defined, coordination consists of protocols, tasks and decision mechanisms designed to achieve concerted actions between and among interdependent units be they organizations, departments or individual actors (Thompson 1967; VandeVen, Delbacq et al. 1976; Kumar and vanDissel 1996). Although there is a promising theory that concentrates on interdependencies between activities (Malone and Crowston 1994), in the reminder of this paper I set out to investigate interdependencies between and among actors (Thompson 1967; Weick 1979) rather than activities because open source projects feature relatively-low task interdependencies due to their modularity. More in detail, the question I attempt to address is the following: how do open source developers organize their interdependencies despite being distributed across space and time? However, as it was outlined in the previous section, coordination in large-scale open source projects is remarkably more interesting and challenging than small-scale ones (Lanzara and Morner 2003). By taking the Linux operating system as a case in point, this paper, therefore, attempts to spell out the way developers manage their interdependencies in large-scale open source projects[2].

3 The Linux case study

Linux is a Unix-like operating system started by Linus Torvalds in 1991 as a private research project. In the early history of the project Torvalds wrote most of the code himself. After a few months of work he managed to create a reasonably useful and stable version of the program and, therefore, decided to post it on a Usenet Newsgroup to get a great number of people to contribute to the project[3]. Between 1991 and 1994 the project size burgeoned to the point that in 1994 Linux was officially released as version 1.0. It is now available free to

[2] In all earnest, I have endeavoured to study only those large-scale projects which fall within the exploration-oriented projects' umbrella. Such projects aim at pushing the frontline of software development collectively through the sharing of innovations. Moreover contributions exist as feedback and are incorporated only if they are consistent with the ideas of the project leader(s). See Nakakoji et al. (2002) on this point.

[3] At a later stage, the Linux code was posted under the GNU/GPL copyright agreement.

anyone who wants it and is constantly being revised and improved in parallel by an increasing number of volunteers (Kollock 1999).

Like many other open source projects (Moody 2001; Weber 2004), Linux exhibits feature freezes from time to time whereby its leader announces that only bug fixes (i.e. corrective changes) will be accepted in order to enhance the debugging process and obtain a stable release version. The Linux kernel development process, therefore, may be decomposed into a sequence of feature freeze cycles each signaling the impending release of a stable version.

Given my concern with coordination processes, I set out to use a longitudinal case study (Pettigrew 1990) as my research design because longitudinal case studies enable researchers to collect data which are processual (i.e. focused on action, as well as structure over time), pluralist (i.e. describing and analyzing the actors' multiple worldviews), historical (i.e. taking into account the emergence of interaction patterns over time) and contextual (i.e. examining wider cultural, institutional and organizational practices that partake in the construction of contexts).

Since February 2002 represents a point of rupture in the lifespan of the Linux development process due to the official adoption of BitKeeper (BK), a proprietary version control tool, by Torvalds, this paper endeavours to analyze the events surrounding the October 2002 feature freeze, the first freeze exhibiting the parallel adoption of two versioning tools, namely BK and CVS (i.e. the Concurrent Versions System), on the assumption that this freeze stands for a test bed for future freezes[4].

In analyzing such events, I looked at communication threads concerning such themes as the patch submission procedure, the bug reporting format, the workflow for incorporating new patches in Torvalds' forthcoming releases, etc. I also studied who interacts with whom in order to spot broader interaction patterns. I decomposed each thread into sets of two contingent responses between two or more perceived others, thus taking Weick's (1979) double interact as my unit of analysis.

To sum up, I took communication cycles of double interacts as my unit of analysis because each double interact refers to an act, a response and a subsequent adjustment between two or more individuals so that each communication cycle can be closed. Although one case study cannot be readily generalized to the universe of open source projects, in dissecting the Linux kernel development interaction patterns I was after analytical generalizations rather than statistical generalizations (Yin 2003).

[4] In early April 2005, Torvalds has replaced BK with Git, a GPL-tool that like BK does not rely on a single, centralized database and maintains a similar workflow for incorporating new patches as discussed below. For more details see: http://www.linux.org/news/2005/04/21/0012.html

4 Coordination processes

If coordination can be conceived of as the management of interdependencies between and among organized actors, how do the Linux kernel developers go about organizing their task interdependencies? My longitudinal case study suggests that there are three coordination processes developers resort to, namely standardization, loose coupling and partisan mutual adjustment.

Standardization encompasses uniform and homogeneous procedures developers enact for problem definition or problem solving (Kallinikos 2004). Such procedures, in turn, refer to the predefined bug reporting format and patch submission routine that developers are to follow whenever they are posting new software misbehaviors (i.e. bugs) or new features on the Linux kernel mailing lists[5]. The Linux documentation archive, for instance, paints a standardized patch submission procedure which expressly mandates that the Linux kernel developers: a) create patches in DIFF format, that is to say in a way that extracts the difference between a modified set of files and an original set of files. The modified version is normally reproduced with a manual procedure called PATCH provided that the original version and the difference are at hand (Yamaguchi, Yokozawa et al. 2000); b) thoroughly describe their changes; c) separate their logical changes into their own patches so that third parties can easily review them; e) carbon copy the LKML (i.e. the Linux Kernel Mailing List); f) submit their patches as plain text or, if they exceed 40 KB in size, provide the URL where they stored their patches; g) provide the name of the version to which their patches apply and include the word PATCH in their subject line[6].

Apart from standardization, loose coupling stands for another critical coordination mechanism that allows developers to manage the workflow for incorporating new patches into Torvalds' forthcoming releases. While standardization pools the developers efforts together and makes them available across spatio-temporal contexts[7], loose coupling coordinates various social subsystems through weak ties.

To exemplify the latter point, consider the following comment posted on the LKML by Linus Torvalds, the chief maintainer, in response to Rob Landley's suggestion to create a "Patch Penguin" to filter incoming software features to Torvalds himself:

[5] The word patch stands for an enhancement or a new feature which is being added to the current kernel.

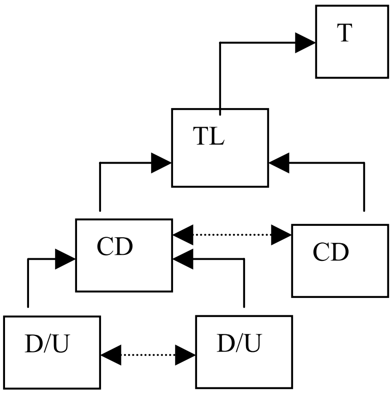
[6] Source: http://lxr.linux.no/source/Documentation/SubmittingPatches. The bug reporting format is also archived in the same repository but the whole procedure has changed after the introduction of Bugzilla, a bug tracking system where all bug reports are stored and automatically tracked down. However, it is worth stressing that, for my purposes, tools as much as manual procedures coordinate pooled interdependencies among a loose network of people by standardizing the process.

[7] The patch submission procedure, for instance, makes individual changes available across spatio-temporal boundaries. By the same token, the bug reporting format allows for reproducing software misbehaviors in space and time.

> Some thinking, for one thing. One "Patch Penguin" scales no better than I do. In fact, I will claim that most of them scale a whole lot worse. The fact is, we've had "Patch Penguins" pretty much forever, and they are called subsystem maintainers. They maintain their own subsystem, i.e. people like David Miller (networking), Kai Germaschewski (ISDN), Greg KH (USB), Ben Collins (firewire), Al Viro (VFS), Andrew Morton (ext3), Ingo Molnar (scheduler), Jeff Garzik (network drivers) etc etc… A word of warning: good maintainers are hard to find. Getting more of them helps, but at some point it can actually be more useful to help the _existing_ ones. I've got about ten-twenty people I really trust, and quite frankly, the way people work is hard-coded in our DNA. Nobody "really trusts" hundreds of people. The way to make these things scale out more is to increase the network of trust not by trying to push it on me, but by making it more of a _network_, not a star-topology around me. In short: don't try to come up with a "Patch Penguin". Instead try to help existing maintainers, or maybe help grow new ones. THAT is the way to scalability.[8]

Although Landley was pushing for an artificial design intervention aimed at institutionalizing a new figure in the Linux development process, namely the "Patch Penguin", Torvalds' remarks eloquently stress that the only way to make the network scale is to work with a limited number of people who, in turn, "work with their own limited number of people"[9]. The upshot of this process is a different way of organizing activities that resembles a heterarchy rather than a hierarchy where the workflow is "kind of a star [shape], with Linus in the center, surrounded by a ring of lieutenants, and these lieutenants surrounded by a ring of flunkies, may be the flunkies surrounded by a ring of flunkies' flunkies, but the flow of the information is through this star, and there are filters. So that you end up with Linus getting stuff that most of the time he doesn't have to work on very hard, because somebody he trusts has already filtered it" (Moody 2001: 179). This workflow is illustrated below:

Figure 1: The Workflow For Incorporating New Patches



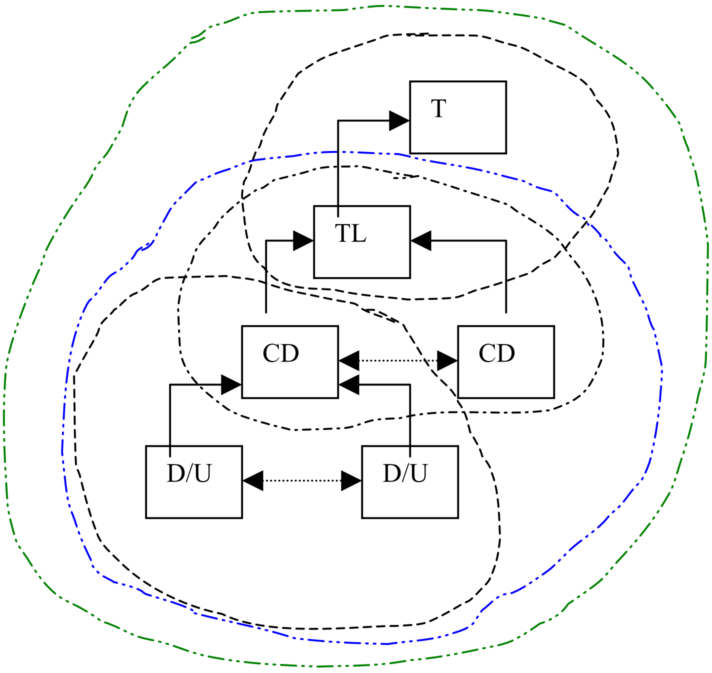
Legend: T=Torvalds; TL=Trusted Lieutenant; CD=Credited Developer; D/U= Developer/User

[8] Source: http://www.ussg.iu.edu/hypermail/linux/kernel/0201.3/1070.html
[9] Source: http://www.ussg.iu.edu/hypermail/linux/kernel/0201.3/1513.html

What are heterarchies? Heterarchies are nested hierarchies (Jen 2002) where the concept of hierarchy does not mean official channels or chains of command from the top down; "[i]nstead, in this context, hierarchy means only that subsystems can differentiate into further subsystems and that a transitive relation of containment within containment emerges" (Luhmann 1995: 19)[10]. Put differently, heterarchies can be conceptualized as loosely-coupled systems considering that, since each system is a part of the whole, as well as being a whole in its own right (Mitleton-Kelly 2003), one can envisage an ensemble of systems where the coupling between systems is weak or loose because the interactions between systems are less direct and less frequent than those within systems (Orton and Weick 1990; Beekun and Glick 2001)[11].

Another way to grasp this idea is to think of loosely-coupled systems as less richly-connected networks where perturbations are slow to spread and/or weak while spreading (Weick 1976) considering that they exhibit weak ties between their subsystems and strong ties within them (Simon 1969; Luhmann 1995)[12] as it is shown below:

Figure 2: Heterarchies as Loosely-Coupled Systems



Legend: T=Torvalds; TL=Trusted Lieutenant; CD=Credited Developer; D/U= Developer/User

[10] Jen (2002: 4) maintains that heterarchies are "interconnected, overlapping, often hierarchical networks with individual components simultaneously belonging to and acting in multiple networks, and with the overall dynamics of the system both emerging and governing the interactions of these networks".

[11] It is worth stressing that, by definition, the degree of connectivity in loosely-coupled systems changes over time. This, in turn, implies that parts that were loosely coupled might become more tightly coupled and vice versa. The crucial point, however, is that these parts always belong to greater wholes which, in turn, are only weakly interconnected with most other parts of their even larger wholes.

[12] Simon (1969) has given this possibility the stature of a hypothesis that he calls the "empty world" hypothesis according to which one should witness higher frequency of interaction within subsystems (i.e. strong ties) than between subsystems (i.e. weak ties).

Despite the trade off between exploration and exploitation (Weick 1979; March 1991; Levinthal 1997), loosely-coupled systems are viable alternatives to such a trade off because they are able to search the space of possibilities through localized adaptations while maintaining local stabilities which ignore limited perturbations elsewhere in the system (Glassman 1973).

4.1 Partisan mutual adjustment and structuring

If the Linux kernel developers resort to coordination by standardization and loose coupling to organize their pooled and sequential interdependencies respectively, how do they go about coordinating their networked interdependencies? Compared to team interdependence where work is fed back and forth within a team of individuals acting jointly and simultaneously (VandeVen, Delbacq et al. 1976), networked interdependence refers to a potentially-unbounded set of developers who work disjointly and asynchronously due to the information and communication technologies (ICTs) in use, as well as other full-time commitments.

I submit that networked interdependencies are coordinated in accordance with the principle of emergence insofar as coordination is the by-product of ordinary decisions undertaken in the pursuit of local interests (Lindblom 1965; Warglien and Masuch 1995). Put differently, whenever developers are disjointly and asynchronously feeding their work back and forth among a potentially-unbounded set of actors, coordination hinges upon the emergence of relatively-stable social structures which are embedded in the mailing lists themselves.

Consider, for instance, the following table outlining the interaction patterns concerning a specific module of the Linux kernel development process:

[Insert Table 1 Here]

The table refers to threads (i.e. communication cycles) that were started on the LKML during the period spanning from October 2002 through December 2002. I have discarded those threads having a single message and/or involving just one developer because I was interested in the frequency of interactions between at least two developers. To illustrate, if one takes the column dated 11/11 entitled “kexec for version 2.5.47”, she should find out a total number of messages equal to 26 where Biederman, the Informal Kexec Maintainer, made 12 postings, Almesberger 3 postings, Pfiffer 2 postings and the perceived others 9 postings 2 of which were made by Bhattacharya, a developer whose involvement with the kexec system call has grown over time[13]. What do these data tell us?

[13] I encompassed all those developers that randomly contributed to the kexec system call under the label “Perceived Others” to stress the potentially-unbounded size of the network. Likewise, Crowston

First, Biederman operates as the Informal Maintainer of the kexec system call because he has started the project and keeps updating it as new kernel versions become available (for instance, kexec for 2.5.42, kexec for 2.5.44, etc.). This point is not trivial: in this project there is an individual who is fully committed because of a public and yet irreversible choice (Weick 1993)[14]. His decision, in other words, is the expression of a free choice of developing or maintaining a patch set which is repeatedly posted on a mailing list for the noticing of others. Hence, there is a choice (i.e. a decision) which is public and irreversible because this patch set is repeatedly updated on the LKML by its Informal Maintainer. This logic, in turn, implies that the Linux kernel developers can take up maintenance roles within the community that feature three distinguishing traits, namely openness (i.e. they are free to develop whatever patch set is of their interest), irreversibility (i.e. once they have started working on a certain patch set, they cannot go back because it is in their interest to keep it in sync with the kernel whether their patch is in the main kernel or not) and visibility (i.e. the other kernel developers know that these people operate as the Informal Maintainers of certain patch sets). Not only can Informal Maintainers start their own patch sets; they can even make variants of Torvalds' releases (Axelrod and Cohen 1999) which co-evolve in a "friendly rivalry"[15] and spur long-term adaptability through better search of the space of possibilities (Stark 2001; Iannacci 2003)[16].

Second, commitment creates a social network around the Informal Maintainer which is open to contributions from everybody. Yet despite being open, the network features quite strong ties among few developers because of their frequent interactions. More in detail, the kexec social network features strong ties among Biederman, Pfiffer and Almseberger. These relationships of networked interdependence can somehow be outlined in the following fashion:

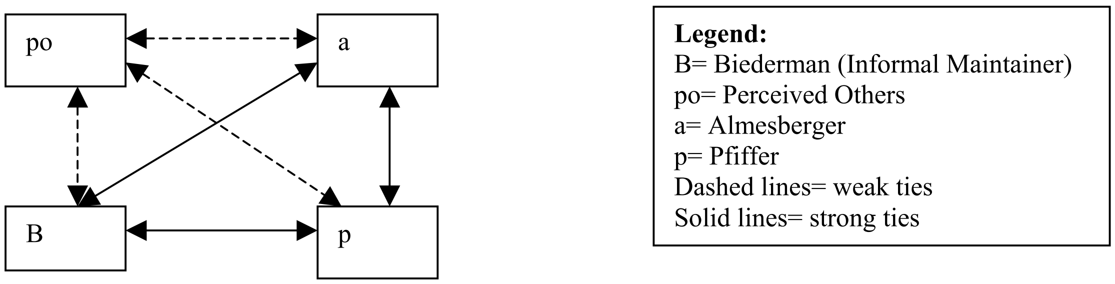
and Howison (2005) have found out a large percentage of messages posted by non-logged-in users that they have labelled as "Nobody".

[14] On the relation between number of messages posted over the electronic medium and commitment see also Sproull and Kiesler (1991: 85-86) where it is argued that the number of messages sent by a contributor is a valid indicator of her degree of commitment.

[15] Source: http://www.ussg.iu.edu/hypermail/linux/kernel/0307.1/0920.html

[16] Noteworthy among these variants are the stable versions released by other trusted lieutenants. Additionally, it is worth stressing that, although Torvalds is in charge of the kernel, which is architecture independent, as well as of the Intel x86 platform, several corporations have started releasing architecture-dependent branches suitable for PPC, Sparc, Sparc64, etc.

Figure 3: Kexec Social Network

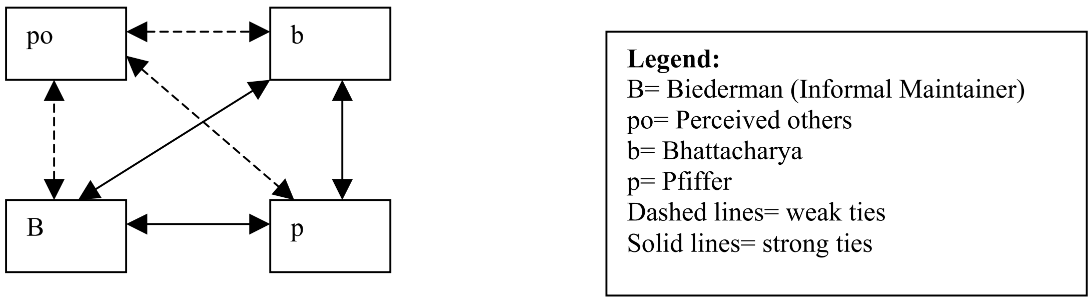


Third, and finally, far from being a constant, the strength of these ties is a variable because the frequency of interactions is bound to change over time as the developers' interests evolve. For instance, despite having interacted quite frequently with Pfiffer and Biederman in the time span under investigation, Almesberger's interests or workload changed over time to the point that his last posting on the LKML dates back to April 3rd, 2003 without involving the kexec system call[17]. Bhattacharya, on the contrary, was quite disenchanted by the kexec system call at first but became gradually more involved with the project in the year 2003 to the point of turning into a stable point of reference for the Informal Maintainer:

> ...Suparna [Bhattacharya] this should be a good base to build the kexec on panic code upon. Until I see it a little more in action this is as much as I can do to help.
>
> And if this week goes on schedule I can do an Itanium port...
>
> Eric [Biederman][18]

This logic, in turn, implies that the by-product of the Linux kernel developers' interactions can be described as a social structure made up of bundles of dyadic ties (Barley 1990) which are bound to evolve as the frequency of the interactions changes. Thus, the kexec social structure evolved over time into a new social network which can be outlined in the following fashion:

Figure 4: Kexec New Social Network



[17] Source: http://www.uwsg.iu.edu/hypermail/linux/kernel/0304.0/0546.html

[18] Source: http://www.ussg.iu.edu/hypermail/linux/kernel/0301.0/1326.html

This last point is further corroborated by the fact that Torvalds' network of trusted lieutenants itself can be considered as an emergent social structure where the strength of the interpersonal ties is indeed a variable rather than a constant:

> You will never figure that out [i.e., Torvalds' network of trust], it isn't predefined. It reshapes itself on the fly, and is really defined by what is going on at any given time. That said, it's usually possible to figure out how [who] the main maintainers are, and what to send where, just don't hope to ever nail that down in a rigid structure. It's not rigid…"[19]

I submit that in highly-unstructured situations where work is fed back and forth among a potentially-unbounded set of loosely-coupled developers, coordination pivots around these minimal social structures because they stabilise the communication processes by making such interactions more orderly, more predictable and more organized. In other words, should one consider postings occurring in 2003, it is very likely that she will find out that Pfiffer is constantly testing the kexec system call patches that Biederman develops over time.

To reiterate, in situations of networked interdependence, coordination is the by-product of partisan mutual adjustments because, while pursuing their localized interests, developers choose only those few projects that are appealing to them. By so doing they inadvertently come to interact with a select number of programmers with whom they are bound to create relatively-stable ties which make social interactions more orderly, more predictable and more organized over time. In addition, Thompson's (1967) additive hypothesis whereby higher degrees of interdependence call for the additive use of all coordination processes being investigated seems to hold up quite well in the Linux case study with the exception that there is a large decrease in the use of loose coupling and much more reliance on ad hoc structuring as a by-product of partisan mutual adjustment because the former mechanism requires a large number of developers to emerge considering that it is a feature of organized complexity (Simon 1969)[20].

[19] Source: http://www.ussg.iu.edu/hypermail/linux/kernel/0201.3/1984.html. Notice that, as opposed to other Informal Maintainers' networks of trust, the issue of information overload has triggered a loosely-coupled social structure around Torvalds' network of trust, thus creating a hierarchy of steps for incorporating new patches in the latest kernel releases as it was pointed out previously. Put differently, loosely-coupled systems require much less information transmission among their parts than other types of systems do and, therefore, are bound to emerge whenever actors face the problem of information overload. On this point see Simon (1969).

[20] To grasp this point, simply look back at the legend of table 1 where you will see [Patch], [CFT i.e. Call for Testing] as standardized procedures for patch submission and bug reporting but you will not see the loosely-coupled workflow for incorporating new functionalities into Biederman's releases, my point being that this is the case because of the relatively-small size of Biederman's network.

5 Conclusions

This paper aims at dispelling the issue of coordination in large-scale open source projects. There are three main contributions that can be drawn from the aforementioned analysis. First, although the electronic means of communication are context-reliant media (Sproull and Kiesler 1991; McKenny, Zack et al. 1992), my analysis shows that, to some extent, the Linux kernel developers are able to reproduce the missing context by resorting to such standardized procedures as the patch submission and the bug reporting routine. Is this a startling finding? The media literature has so far maintained that electronic contexts enhance the problem of equivocality, thus requiring face-to-face communications as complementary media (Daft and Lengel 1986; Nohria and Eccles 1992). My reading of the Linux case study questions this argument by showing that what is needed is a structural substrate whereupon social interaction can occur. This, in turn, means that the real issue is a problem of *grade or minimal threshold*: once developers resort to uniform and homogeneous procedures for problem definition and problem solving, they can reproduce the missing context *to a certain degree*. At this point, rich communication media are no longer required because developers can interact across spatio-temporal boundaries despite the fact that the electronic media in use filter out the social context cues.

Second, although Van de Ven et al. (1976) have already felt the need to extend Thompson's (1967) concept of reciprocal interdependence by introducing the notion of team interdependence, my reading of the Linux case study shows that their influential notion needs further extension: while team interdependence refers to workflows where "the work is acted upon jointly and simultaneously by unit personnel at the same point in time" (Van de Ven et al. 1976: 325), I submit that a new concept needs to be entertained by organization scholars, namely the idea of *networked interdependence* featuring a potentially-unbounded team where work is performed in a disjoint and asynchronous fashion between various full-time task assignments.

Third, and finally, networked interdependence is coordinated through a *structuring process* which is the by-product of partisan mutual adjustments rather than a separate and centralized set of coordinating decisions (Lindblom 1965). To reiterate, while pursuing their localized interests, developers coalesce around few projects that are appealing to them. By so doing they inadvertently come to interact with a select number of programmers with whom they are bound to create relatively-stable ties on the basis of decisions which are public and relatively-irreversible because of the high frequency of interactions. I submit that these emergent structures operate as pivotal *coordination mechanisms* because they make social interactions more orderly, more predictable and better organized over time. Ultimately, my claim is that "genuine parallelism" (Warglien and Masuch 1995: 4) is the distinguishing feature of the

Linux kernel development process where truly-independent parallel search (Cohen 1981) by multiple developers walking many directions at once (Iannacci 2003) is corroborated by the emergence of relatively-small social clusters which attend to specific areas of the source code on the basis of relatively-stable social ties.

References

Axelrod, R. M. and M. D. Cohen (1999). Harnessing complexity: organizational implications of a scientific frontier. New York, Free Press.

Barley, S. R. (1990). "The Alignment of Technology and Structure through Roles and Networks." Administrative Science Quarterly **35**(1): 61-103.

Beekun, R. I. and W. H. Glick (2001). "Organization Structure from a Loose Coupling Perspective: A Multidimensional Approach." Decision Sciences **32**(2): 227-251.

Benkler, Y. (2001). Coase's Penguin, or, Linux and the Nature of the Firm. Conference on the Public Domain, Durham, North Carolina.

Cohen, M. D. (1981). "The Power of Parrallel Thinking." Journal of Economic Behavior and Organization **2**(4): 285-306.

Crowston, K. and J. Howison (2005). "The Social Structure of Free and Open Source Software Development." First Monday 10/2; Address: http://firstmonday.org/issues/issue10_2/crowston/index.html Accessed on: 10/04/05.

Daft, R. L. and R. H. Lengel (1986). "Organizational Information Requirements, Media Richness and Structural Design." Management Science **32**(5): 554-571.

Feller, J. and B. Fitzgerald (2000). A Framework Analysis of the Open Source Software Development Paradigm. Proceedings of the Twenty-First International Conference on Information Systems, Brisbane, Australia.

Glassman, R. B. (1973). "Persistence and loose coupling in living systems." Behavioral Science **18**: 83-98.

Healy, K. and A. Schussman (2003). "The Ecology of Open Source Software Development." MIT Site; Address: http://opensource.mit.edu/papers/healyschussman.pdf Accessed on 03/10/03.

Iannacci, F. (2003). "The Linux Managing Model." First Monday 8/12; Address: http://www.firstmonday.org/issues/issue8_12/iannacci/index.html, Accessed on 03/12/03.

Jen, E. (2002). "Stable or Robust? What's the Difference?" Santa Fe Institute; Address: http://www.santafe.edu/~erica/stable.pdf Accessed on 16/08/04.

Kallinikos, J. (2004). "The Social Foundation of the Bureaucratic Order." Organization **11**(1): 13-36.

Kollock, P. (1999). The Economies of Online Cooperation: Gifts and Public Goods in Cyberspace. Communities in Cyberspace. M. Smith and P. Kollock. London, Routledge.

Kumar, K. and H. G. vanDissel (1996). "Sustainable Collaboration: Managing Conflict in Interorganizational Systems." MIS Quarterly **20**(3): 279-300.

Lanzara, G. F. and M. Morner (2003). "The Knowledge Ecology of Open-Source Software Projects." 19th EGOS Colloquium; Address: http://opensource.mit.edu/papers/lanzaramorner.pdf, Accessed on 07/07/2003.

Levinthal, D. A. (1997). "Adaptation on rugged landscapes." Management Science **43**(7): 934-950.

Lindblom, C. (1965). The Intelligence of Democracy. New York, Free Press.

Luhmann, N. (1995). Social Systems. Stanford, California, Stanford University Press.

Malone, T. W. and K. Crowston (1994). "The interdisciplinary study of coordination." ACM Computing Surveys **26**(1): 87-119.

March, J. G. (1991). "Exploration and Exploitation in Organizational Learning." Organization Science **2**(1): 71-87.

McKenny, J. L., M. H. Zack, et al. (1992). Complementary Communication Media: A Comparison of Electronic Mail and Face-to-Face Communication in a Programming Team. Networks and Organizations: Structure, Form, and Action. N. Nohria and R. G. Eccles. Boston, Massachusetts, Harvard Business School Press.

Mitleton-Kelly, E. (2003). Ten Principles of Complexity and Enabling Infrastructures. Complex systems and evolutionary perspectives of organisations: the application of complexity theory to organisations. E. Mitleton-Kelly. Oxford, Pergamon, 2003.

Moody, G. (2001). Rebel Code: Linux and the Open Source Revolution. Hamondsworth, Middlesex, England, Allen Lane, The Penguin Press.

Muffatto, M. and A. Faldani (2003). "Open Source as a Complex Adaptive System." Emergence **5**(3): 83-100.

Nakakoji, K., Y. Yamamoto, et al. (2002). Evolution patterns of open-source software systems and communities. International Workshop Principles of Software Evolution.

Nohria, N. and R. G. Eccles (1992). Face-to-Face: Making Network Organizations Work. Networks and Organizations: Structure, form, and Action. N. Nohria and R. G. Eccles. Boston, Massachusetts, Harvard Business School Press.

Orton, J. D. and K. E. Weick (1990). "Loosely Coupled Systems: A Reconceptualization." Academy of Management Review **15**(2): 203-223.

Pettigrew, A. M. (1990). "Longitudinal field research on change: theory and practice." Organization Science **1**(3): 267-292.

Raymond, E. S. (1999). The Cathedral and the Bazaar. Sebastopol, CA, O'Reilly.

Simon, H. (1969). The Architecture of Complexity. The Sciences of the Artificial, The Riverside Press, Inc.

Sproull, L. and S. Kiesler (1991). Connections: New Ways of Working in the Networked Organization. Cambridge, Massachusetts, The MIT Press.

Stark, D. (2001). Ambiguous Assets for Uncertain Environments: Heterarchy in Postsocialist Firms. The Twenty-First century Firm: Changing Economic Organization in International Perspective. P. DiMaggio. Princeton, Princeton University Press.

Thompson, J. D. (1967). Organizations in Action. New York, Mc Graw-Hill Book Company.

VandeVen, A. H., A. L. Delbacq, et al. (1976). "Determinants of Coordination Modes Within Organizations." American Sociological Review **41**(2): 322-338.

von-Hippel, E. (2001). "Open Source Shows the Way: Innovation by and for Users - No Manufacturer Required!" MIT Site; Address: http://opensource.mit.edu/papers/evhippel-osuserinnovation.pdf, Accessed 10/11/02.

Warglien, M. and M. Masuch (1995). Introduction. The Logic of Organizational Disorder. M. Warglien and M. Masuch. Berlin, De Gruyter.

Weber, S. (2004). The Success of Open Source. London, England, Harvard University Press.

Weick, K. E. (1976). "Educational Organizations as Loosely Coupled Systems." Administrative Science Quarterly **21**(1): 1-19.

Weick, K. E. (1979). The social psychology of organizing. Menlo Park, California, Addison-Wesley Publishing Company.

Weick, K. E. (1993). Sensemaking in organizations: Small structures with large consequences. Social Psychology in Organizations: Advances in Theory and Research. K. Murnigham. Englewood Cliffs, NJ, Prentice-Hall.

Yamaguchi, Y., M. Yokozawa, et al. (2000). Collaboration with lean media: how open-source software succeeds. Proceedings of the Conference on Computer Supported Cooperative Work (CSCW '00), Philadelphia, PA.

Yin, R. K. (2003). Case Study Research: Design and Methods. Thousands Oaks, Sage Publications.

List of tables and figures:

Table 1: Kexec System Call Interactions[21]

	17/10	18/10	19/10	23/10	06/11	07/11	11/11	17/11	01/12	14/12	16/12
Perceived Others (Bhattacharya)	-	5	6	5	1	9	9 (2)	8 (2)	2	4	1 (1)
Pfiffer	-	5	1	-	1	3	2	3	-	-	2
Almesberger	1	2	-	-	-	3	3	1	-	-	-
Biederman	1	14	9	1	3	9	12	12	3	4	3

Feature Freeze

Legend:
Developers:
Biederman, Almsesberger and Pfiffer (Network of trust)
Perceived others (Other developers partaking in the dialogic negotiations)
Hellwig: Member of LSM network of trust
Bhattacharya: Future member of kexec network of trust

Time:
17/10 Thread Subject Line: “kexec for 2.5.42”
18/10 Thread Subject Line: “[CFT] Kexec syscall for 2.5.43”
19/10 Thread Subject Line: “kexec for 2.5.44”
23/10 Thread Subject Line: “Heary AOL for kexec”
06/11 Thread Subject Line: “kexec for 2.5.46”
07/11 Thread Subject Line: “kexec (was [lkcd-devel] Re: what’s left over)”
11/11 Thread Subject Line: “kexec for v. 2.5.47”
17/11 Thread Subject Line: “[Announce] kexec-tools-1.6 released”
01/12 Thread Subject Line: “[Announce] kexec-tools-1.8”
14/12 Thread Subject Line: “[Patch] Kexec for 2.5.51”
16/12 Thread Subiect Line: “[Patch] kexec for 2.5.52”

[21] Kexec is a system call that allows to load a kernel from the currently executing Linux kernel. Source: http://www.ussg.iu.edu/hypermail/linux/kernel/0210.2/1065.html