COORDINATION BY REASSIGNMENT IN THE FIREFOX COMMUNITY

Matthijs den Besten
Groupe Sup de Co Montpellier, Montpellier Research in Manangement, Montpellier, France, matthijs.denbesten@gmail.com

Jean-Michel Dalle
CRG - Ecole Polytechniqü, Palaiseau, France, jean-michel.dalle@upmc.fr

Matthijs den Besten and Jean-Michel Dalle, 2014, "COORDINATION BY REASSIGNMENT IN THE FIREFOX COMMUNITY", Proceedings of the European Conference on Information Systems (ECIS) 2014, Tel Aviv, Israel, June 9-11, 2014, ISBN 978-0-9915567-0-0
http://aisel.aisnet.org/ecis2014/proceedings/track17/6

COORDINATION BY REASSIGNMENT IN THE FIREFOX COMMUNITY

Complete Research

Den Besten, Matthijs, Groupe Sup de Co Montpellier Business School Montpellier Research in Management & CRG (CNRS and Ecole Polytechnique), Montpellier, France, m.den-besten@supco-montpellier.fr

Dalle, Jean-Michel, University Pierre et Marie Curie (UPMC), Paris, France, jean-michel.dalle@upmc.fr

Abstract

According to the so-called "mirroring hypothesis", the structure of an organization tends to replicate the technical dependencies among the different components in the product (or service) that the organization is developing. An explanation for this phenomenon is that socio-technical alignment, which can be measured by the congruence of technical dependencies and human relations (Cataldo et al., 2008), leads to more efficient coordination. In this context, we suggest that a key organizational capability, especially in fast-changing environments, is to quickly reorganize in response to new opportunities or simply in order to solve problems more efficiently. To back up our suggestion, we study the dynamics *of congruence between task dependencies and expert attention within the Firefox project, as reported to the Bugzilla bug tracking system. We identify in this database several networks of interrelated problems, known as "bug report networks" (Sandusky et al., 2004). We show that the ability to reassign bugs to other developers within each bug report network does indeed correlate positively with the average level of congruence achieved on each bug report network. Furthermore, when bug report networks are grouped according to common experts, we find preliminary evidence that the relationship between congruence and assignments could be different from one group to the other.*

Keywords: Conway's Law, mirroring hypothesis, congruence, open-source software, bug report network, collaboration-based innovation

1 Introduction

Besides the many reasons that tend to slow or even hinder their capabilities to adapt, organizations are in a sense constrained by their own products – or conversely it is the products, which are constrained by organizational architecture. As Conway (1968) famously wrote in his seminal article, organizations are "bound to replicate the architecture of the very product or service they produce" (Conway, 1968). This hypothesis, of a more general coupling between organization and product, initially known as Conway's Law, and now as the "mirroring hypothesis", has spurred strong recent interest in the

management and information systems literature (e.g. Coplien, 2006; Amrit & van Hillegersberg, 2008).

Understanding how mirroring occurs is indeed key to designing more efficient organizations and information systems. However, recent literature has focused mainly on coarse-grained architectural features – first and foremost, on modularity –, but a finer-grained understanding of how small changes at the micro level actually add up to produce macro evolutions is still needed. This understanding needs to be dynamic, for the *alignment* of expertise with the technical requirements of a collection of interdependent issues stands out as a logical candidate to explain micro-characteristics of organizational flexibility, and thus is bound to be dynamic by its very nature, since assigning and reassigning problems to various individuals is a dynamic process subject to many and regular changes.

The relative scarcity of academic research on the micro-characteristics of mirroring is in part due to the lack of appropriate datasets and of measurement methodologies. With respect to measurement, an important step was made by Herbsleb and colleagues, who introduced the notion of "socio-technical congruence" and suggested ways to measure it (Cataldo *et al.,* 2008). With respect to datasets, the traces of activities of developers involved in the creation and maintenance of open source software provide a source waiting to be exploited in full (Crowston et al., 2006; Kwan *et al.*, 2012).

In this paper, we study a dataset from an online archive maintained by the Mozilla foundation through a key element of its information system, the issue tracker Bugzilla: Bugzilla helps to store reports of "bugs" with the software it maintains and develops, specifically here with respect to the well-known Firefox browser, and to record actions undertaken to resolve issues that arise. We automatically detect dependencies among tasks and study dynamically how they "congrue", thanks to links between developers. We explore the relationships between congruence, measured at the level of sub-groups of interrelated problems, previously named bug-report networks (Sandusky et al., 2004), and the propensity of tasks to be re-assigned by developers. In this way, we relate the micro-adaptability practices of the Firefox community to its macro-capability to dynamically maintain congruence between technical dependencies and human relationships.

We first review relevant literature; we then describe the original methodology we have applied to analyze our dataset and finally present and discuss our findings.

2 Background

The idea that the structure of an organization affects the technical structure of the product or service that it develops and vice-versa, sometimes called the "mirroring hypothesis", was initially proposed by Conway (1968). Conway went as far as stating that organization and product architecture were "homomorphic." Recent interest in the literature has been revived by MacCormack, Rusnak, and Baldwin (2006)'s study of modularity, which convincingly argued that the software source code base of Netscape had to be drastically re-organized in order to adapt to the new mode of organization that was implied by move in 1998 to confide the development and maintenance of the Netscape browser to the Mozilla foundation.[1] MacCormack et al. (2006) showed that the code had to be made more modular in order to allow for the large scale distributed development model of open source software that the Mozilla foundation pursued. Building upon this idea, MacCormack, Baldwin and Rusnak (2012) further established for several open source projects that their structure in terms of code dependencies was more modular than the structure of equivalent closed source projects.

[1] See also Langlois & Garzarelli (2008) on the impact of modularity on organizational form.

The *macro* benefits of modularization, however, rest upon the *micro*-alignment of communications and technical dependencies, as dependencies between problems need to be addressed. Modularity not only reduces the span of technical dependencies through information hiding (Parnas, 1972): modularization also results in congruence at the architectural level (Cataldo et al., 2008; Cataldo et Herbsleb, 2013), and modularity finally builds upon congruence at the level of tasks (Kwan, Cataldo, and Damian, 2012): developers typically face a set of tasks assigned to them and they need to coordinate whenever there are dependencies among these task sets, which can happen for instance when two or more tasks affect the same file or function in the source code base. Congruence is therefore inherently dynamic since technical dependencies change due to product evolution, sometimes even at the architectural level.

Transparency, increasing awareness of the actions of others, has recently been suggested as a general solution to this dillema (Dabbish et al., 2013; Colfer & Baldwin, 2010), since it lowers coordination costs by reducing the need for explicit communication. As technical dependencies are made explicit, developers can develop separately and solve coordination issues implicitly. Transparency, however, is certainly not sufficient in itself. First, transparency results in information overload, which makes it necessary to develop and maintain metadata information used for filtering (den Besten and Dalle, 2008; Tsay et al., 2013), probably dependent upon shared mental models within the community. Yet, when communication is reduced to listening to and following these self-selected metadata signals, it is not certain that congruence can be fully attained. Furthermore, signalling of this kind can give rise to macro-level self-organization phenomena, some of them "stigmergic" (den Besten, *et al.*, 2008; Bolici et al., 2009) affecting the allocation of efforts and thus product design in complex non-linear ways (Dalle & David, 2005). Second, and maybe more crucial with respect to congruence, transparency is structurally limited by and hardly resistant to *problem ownership* – the very explanation indeed put forward by Conway to justify his "law": "As long as the manager's prestige and power are tied to the size of his budget, he will be motivated to expand his organization" (Conway, 1968, p. 31)[2]. Problem ownership is recognized as an issue of importance not only in software engineering (Nordberg, 2003), where it has been a major motive for the development of so-called "agile" development methodologies[3], but also in other collective efforts like Wikipedia (Thom-Santelli et al., 2009), which is consistent with the role that reputation and regard by fellow members have acquired in online communities (Dalle et al., 2004). Problem ownership constrains the adaptability of any kind of organization by not allowing someone to contribute to the solution of a problem owned by someone else – however transparent the environment can be, even if it is "actionably transparent" (Colfer and Baldwin, 2010)[4] – and therefore constrains its congruence or more precisely its ability to "congrue" dynamically.

How, then, is congruence achieved dynamically under the constraints of problem ownership? Why, then, would some environments, especially in online communities, be more "actionable"? We suggest that the latter could structurally be associated with a more *limited* ownership of problems – with a more *open* attitude towards problem ownership, which is not unrelated to Raymond's intuitive conception of "Lockean property rules" (Raymond, 2001) within open-source software communities.

---

[2] It is thus very surprising that the most recent survey of Conway's Law does not mention problem ownership (Bailey et al., 2013).

[3] West and O'Mahony (2008) suggested that successful open source development requires tools for participation.

[4] Indeed, if a design is actionably transparent, "there is no need to make [a] conjecture comprehensible to another person or persuade someone else that a new idea is worth trying" (Colfer and Baldwin, 2010, p. 23). Actionable transparency therefore reduces the needs for coordination. However, perfect actionability is an uncommon situation (Colfer and Baldwin, 2010, p. 22: "In practice, perfect actionability is rare: almost always, some controls are in place regarding who can change the master design.") particularly limited by problem ownership issues.

Without such an attitude, it is difficult to achieve "dynamically assigned ownership" Nordberg (2003): situations where problem ownership rules would not hinder the dynamic reassignment of issues.

In cases where problem ownership is less problematic, complex problems can typically be addressed by more than one person, as was observed in a collection of large open-source projects (den Besten*, et al.,* 2008). Furthermore, clues were found in this study that projects that were not born in open-source environments could not exhibit congruence between team formation and problem dendencies, even years after their code was open-sourced. These results suggest that there exist organizational-level characteristics and capabilities of communities that allow them to address problems more efficiently. If the assignment of problems changes more easily, consequently in such organizations the level of congruence should be generally correlated to the *softness* of problem assignments – that is, to the intensity of problem reassignments. This is the hypothesis that we explore in this paper.

Organizations in which problem assignments would be more amenable to change than is usually the case could be labelled "softs," in order to emphasize the importance of this specific characteristic on their organizational capabilities, and to question the ability of firms to be soft. Indeed, as Conway put it: "flexibility of organization is important to effective design." (Conway, 1968, p. 31) Needless to say, softs are not limited to open-source software communities, as suggested for instance by the large literature on agility (Dingsøyr *et al.,* 2012).[5] Nor is it the case that "softness" is a capability that is evident in all open source communities, as case studies on ZOPE (Feller *et al.,* 2006) or LaTeX (Gaudeul, 2007) attest. We simply suggest that, in softs, problems can be *addressed* more efficiently through more fluid and agile assignments, resulting in an improved dynamic changeability of products: first, because they have an *address* that anyone can find, thanks to transparency, and second, because they can *be addressed* – are actionable – due to softer problem ownership.

3 Data

Kwan *et al.* (2012) point to bug reports as a promising data source in order to study congruence from a task-based perspective. In line with this advice, we study data from Bugzilla, an issue tracking system developed and used by the Mozilla foundation to keep track of development and maintenance problems – also known as "bugs" – related to Mozilla software, such as the well-known browser Firefox. Bugzilla keeps track of all contributions that eventually lead to the resolution of each bug. That is, it keeps track of forum-like discussions, but also of the history of all changes to various other metadata characteristics of each bug, such as its status (UNCONFIRMED, NEW, ASSIGNED, FIXED, CLOSED, VERIFIED), its assignee, etc. All information about each bug – a "bug report"– is public (with the exception of bugs relating to software security which are kept closed to outsiders as long as the issue described in the bug report has not been resolved). Figure 1 provides a screenshot of a bug report.

Bugzilla further maintains metadata in each bug report about the existence of formal links between the reported bug and others. These cross-references are of two types: developers and users reporting a bug or working on the resolution of a bug can indicate that the resolution of other bugs "depends" on this particular bug, or that its resolution is conversely "blocked" by the one of other bugs. Blocking is the logical complement of depending: when bug A is indicated as blocking bug B, most of the time bug B will also be listed as being dependent on bug A. This reciprocity ensures that both people looking at bug report A and those looking at bug report B will be aware of the dependency between them.

[5] On organizational flexibility and "fluidity", and from different perspectives, see also and Glance & Huberman (1993) and Langlois (2007).

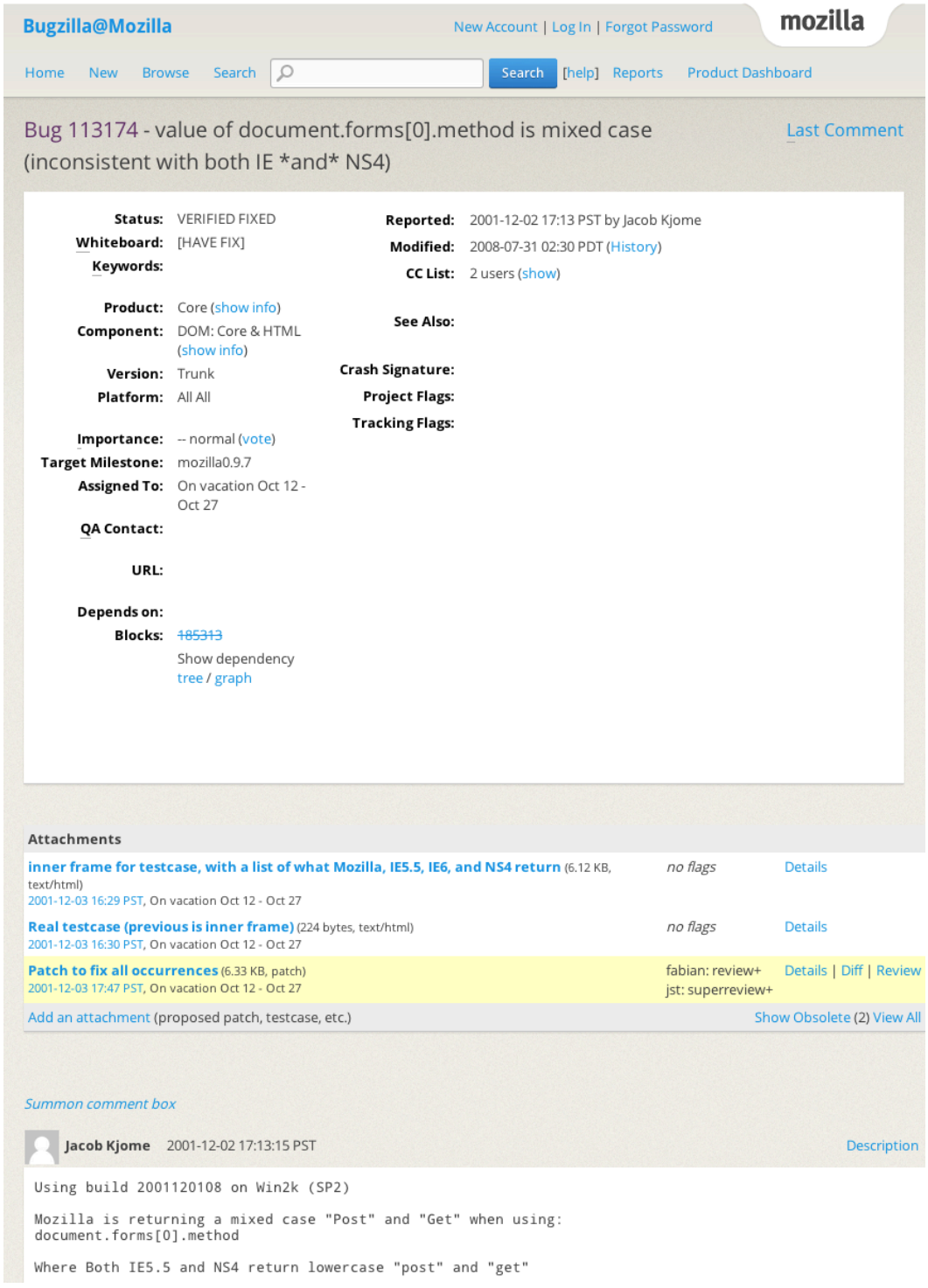

Figure 1. Screenshot of bug report 113174. Shown are bug metadata plus additional information and a description of the bug. Not shown is the ensuing discussion. The strikethrough of the bug listed as blocking indicates that the bug in question has been resolved. It is possible to log on to Bugzilla. In that case links to email addresses are provided for each participant. The text in the field assigned to is the participant

> *handle that is linked to his/her email address. If we had logged in we would be able to access the actual email.*

On the basis of these dependency relationships bugs can be grouped into bug report networks (Sandusky *et al.,* 2004). Indeed, Sandusky (2005) found that mapping out dependencies among problems is an important precursor to problem solving activity on Bugzilla and explored the formal and informal ways through which those networks of bug reports are being constructed.

In issue tracking systems like Bugzilla, *assignment* is the result of the *triage* that a bug undergoes after it has been reported to the system (Villa, 2003). The person who has been assigned the report is then tasked to orchestrate the resolution process. According to Guo et al. (2011) there are five primary reasons for reassignments: "finding the root cause", that is, discovering that the problem is of a slightly different nature and that therefore someone else is better placed to oversee its resolution, "determining ownership", that is, the person who was supposed to take care of the report disappeared and someone else needs to be found, "poor bug report quality", "hard to determine proper fix", that is, more reasons why the "root cause" was not found immediately, and "workload balancing", that is, the person charged with overseeing the resolution had already too much on his or her plate. The right person to oversee the resolution of a problem is likely to be someone who is already familiar with similar problems. Hence, frequent assignments will probably help in aligning tasks as the understanding of the problem and its dependencies improves (just like well-timed short increases in temperature help the process of annealing, cf. Carley and Svoboda (1996)).

Bugs in Bugzilla are assigned a unique identifier meta-data describing the bug. We rely on this feature to sample our data. In particular, we randomly draw about 4000 numbers between 100 000 and 300 000 and retrieved the bug reports with the corresponding bug-id. Identifiers being attributed sequentially (new report gets identifier number of previous report plus 1), this sample contains bugs declared between 2002 and 2005, which covers the period in which Mozilla developers started to work on Firefox until the release of Firefox 1.5.

Among the bugs we retrieved, some were identified in the reports as "tracking bugs" or "meta-bugs": these bugs are mostly "to-do lists" for developers, and therefore do not identify technical dependencies properly speaking. We therefore choose to discard them from our analysis here: specifically, we discarded bug reports in which the word "tracking" appeared in the bug description and those that had "meta" as keyword, or "tracking" as "component" in the bug meta-data. Besides, only bug reports whose current status is identified as "verified" or "resolved" are included in the networks expanded from the initial sample of bugs. The rationale for this restriction is that we want to study the congruence the congruence between technical *systems* and developer activity on those systems. Meta-bugs risk to include relationships of a less or non-technical nature and turn the network into a loose *collection* of bugs in which coordination is consequently less important (Mateos Garcia and Steinmueller, 2003).

To reconstruct bug report networks (BRNs), we retrieved bugs listed as "depends on" and "blocking" for each bug report in our initial sample, and reports describing them as well. We iterate this process until up to ten degrees of distance from each "root" bug report in the initial sample. Our analysis focuses on complete networks. That is, networks, which are no longer growing in size after ten iterations. Among these networks we kept only those with at least 30 bugs since we found that smaller networks contain too few data points to measure congruence. Thus we obtained 11 distinct bug report networks. Two among them contained a bug report with at least 50 dependencies. Among further inspection these bugs appeared to be tracking bugs and so we decided to remove them and with them the bug report networks as their components thus disconnected had sizes much lower than 30. Figure 2 presents a graphical representation of the varied shapes of remaining BRNs. The tree-like structures of the graphs reflect the hierarchical nature of dependency relationships.

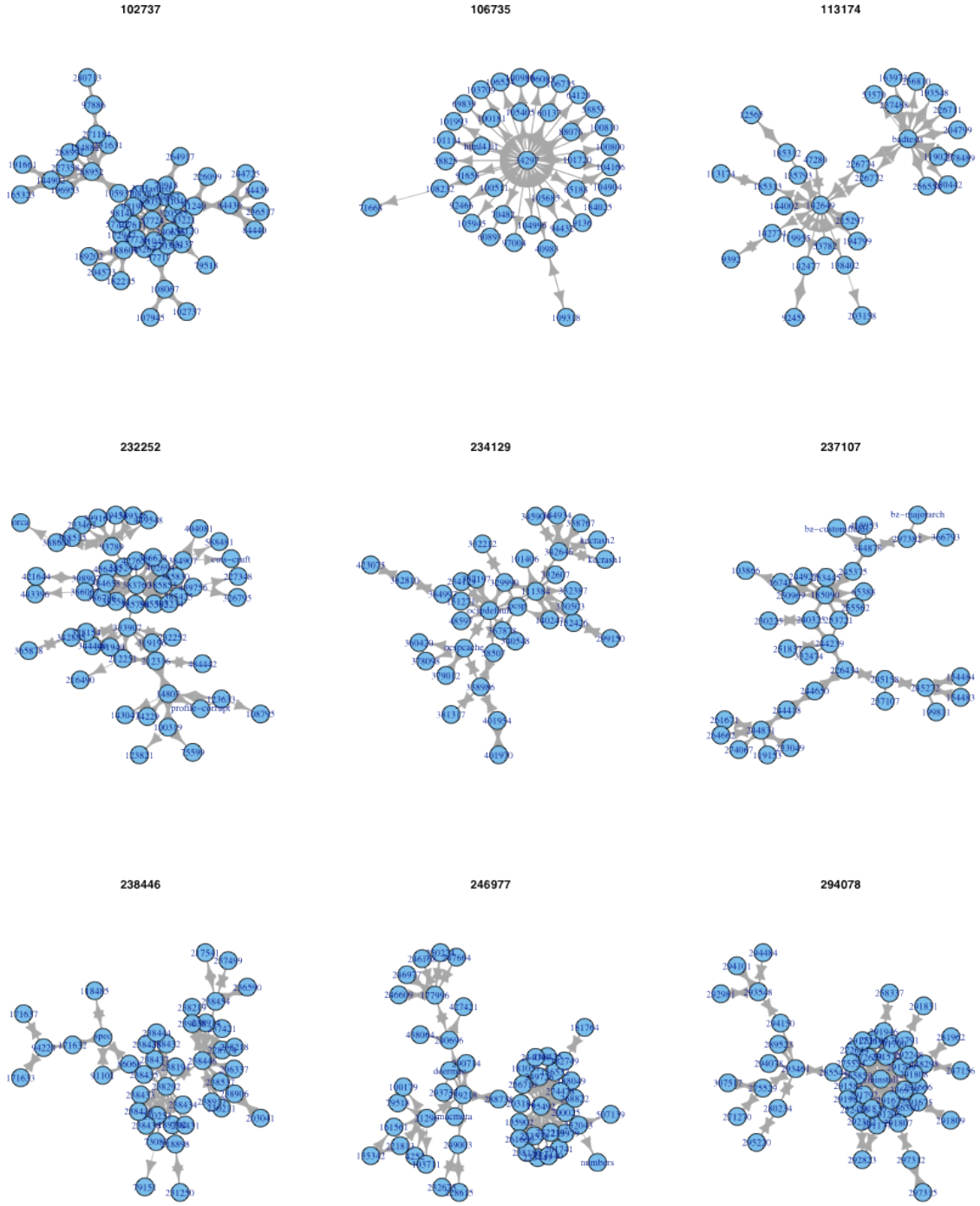

Figure 2. *Graphical representation of the 9 bug report networks selected for analysis. The number above each graph identified the bug-id from which the network was constructed. The arrows indicated the "depends-on" and "blocking" dependencies among bugs.*

BRN	*comments*	*bugs*	*participants*	*frequent participants*	*First comment*	*25% of cmmnts*	*50% of cmmnts*	*75% of cmmnts*	*Last comment*
102737	909	51	172	48	jun-00	Oct-01	fév-03	oct-04	sep-12
106735	627	39	141	43	jul-99	jul-01	oct-01	nov-01	jan-12
113174	357	32	91	32	jul-99	Jan-02	avr-03	nov-03	déc-10
232252	1469	56	408	121	sep-99	mai-03	jan-07	oct-07	nov-12
234129	813	36	100	33	aoû-00	Avr-06	jan-07	jun-07	jan-13
237107	743	35	124	44	fév-00	Déc-02	jul-04	mar-05	déc-12
238446	685	42	149	59	oct-00	Jan-04	mar-04	mai-04	mar-11
246977	1060	52	266	86	mar-99	mai-03	jul-04	déc-04	fév-12
294078	821	51	100	47	jun-04	Avr-05	avr-05	mai-05	aoû-09

Table 1. For each bug report network (BRN), identified by one of its bugs, the number of comments contribution to the bugs' discussion threads, the number of bugs in the network, the number of commentators ("participants"), the number of frequent participants (those who left at least three comments), and the range of dates when the comments were made.

We measure the *congruence* for each BRN on the basis of the dependencies listed in the bug reports[6] and of common contributors. Where Cataldo *et al.* (2008) propose to measure congruence as the ratio between coordination requirements derived from task dependencies and observed coordination activities, we look directly at the *proportion of technical dependencies for which there is also a human link*. As communication on bug reports is *asynchronous* and traces of *synchronous* communication on IRC channels are not systematically archived, it is hard to establish a communication link. Yet, it is relatively trivial to establish whether two bugs have received attention of the same person at a particular point in time: we consider that a human resource link between two bug reports is established when there is at least one person who is active on both bug reports in a sliding window of 100 actions,[7] activity being understood in a broader sense to include comments contributed to the bug's discussion thread as well as automatically generated comments linked to other actions on the bug report such as the declaration of a dependency or the submission of a patch, etc. In order to match people we match the email-addresses with which they identified themselves to Bugzilla (after removal of strings of characters between "+" and "@" as those are ignored by email dispatchers as well). By averaging congruence over sliding 100 action windows, we characterize how each BRN is able to dynamically maintain a given level of congruence. Besides, we measure assignment changes for each BRN by

[6] In practice, these dependencies tend to be discovered and edited as the discussion about bug resolution proceeds; the dependency list in the report is updated accordingly. Because the bug reports in our sample are old, and as the bugs described in the report have been resolved, it seems safe to assume that the technical dependencies listed record actual dependencies.

[7] Some of the discussions span very long periods and/or contain many comments. We do not want to assume that a developer who has contributed once to a discussion is paying attention to the associated bug during the complete life cycle of that bug.

counting how many times the "assigned to" field was changed during similar 100 action windows and compute its dynamic average to characterize the fluidity of assignments for each BRN.

4 Results

Table 1 provides descriptive statistics, including indicators of the scale and scope, for bug report networks. Note that time between the first comment to a bug in the network and the last comment contributed in that network often spans many years. Nevertheless, most comments are posted within months. Typically there are many people who contribute to the discussion threads of the bugs in the networks, but most of them comment only once or twice.

There is no overlap among networks in terms of bugs. In terms of participants, matching of email-addresses of the contributors to the discussion threads reveals that there is relatively little overlap. On average two networks have 22 participants in common. Among them, 7 on average contributed more than two comments in both networks. On the basis of the participation identities it is possible to split the networks into three distinct groups through application of hierarchical clustering using Ward's method on a distance matrix defined by the one-complement of the Jaccard similarity (Levandowsky and Winter, 1971) of sets of participants associated with the networks. The different groups – cluster *A* with networks around bugs 102737, 106735 and 113174; cluster *B* with networks around bugs 232252, 238446, 246977, 294078; and cluster *C* with networks around bugs 234129 and 237107 –roughly correspond to different periods in time. This suggests that the differences in the identities of frequent participants are due to turnover within the Mozilla/Firefox community. At the same time different networks deal with different problems and so attract different experts.

Figure 3 plots the average congruence and the average number of assignments per 100 comments for the selected bug report networks, distinguishing between the 3 different clusters that defined above. The distribution of the data points around the diagonal suggests that these two numbers are related, as hypothesized above. The correlation coefficient for average congruence and average number of assignment changes is 0.89 with $p = 0.0011$. In order to check the robustness of this result, we also computed the correlation coefficients on vectors with one of the networks left out. The estimated coefficient is stable around 0.9 for these 8 permutations and the corresponding p-values range from less than 0.001 if network for bug 106735 is left out and 0.005 if network 113174 is left out. The three clusters of networks grouped according to participant commonality also nicely split in terms of the average level of congruence attained. It also appears that the levels of congruence attained is higher for the cluster with the oldest bugs, group A, than for the other clusters. A potential interpretation for this finding could be related to a gradual "hardening" of the organization as Firefox established itself and/or to a change in the ownership structure as Firefox matured.

5 Discussion and conclusion

The results presented in this paper, based on a study of the dynamics of collaboration-based innovation in the context of the Firefox open source project, and more specifically on an analysis of development activities logged by the Bugzilla issue tracker system, provide support to the hypothesis that flexibility in assignments influences the capability of an organization to maintain congruence between technical and work dependencies. A confirmation is now needed, on a larger sample of BRNs, over a longer period of time, for different forms of bug dependencies and for other open source projects. In addition, the role of tracking bugs deserves deeper analyses, and also the fact that congruence and re-assignment frequency might change (or not) over time. Furthermore, the role of Bugzilla as a component in the information system of the community has also to be clarified.

If they were confirmed, these results would suggest that flexibility in problem ownership is a way to maintain a dynamic alignment between technical and work dependencies and thus to improve

congruence and performance. It would plead for both organizational and information systems solutions to make problem reassignment "cheaper", and hence organizations softer and more fluid.

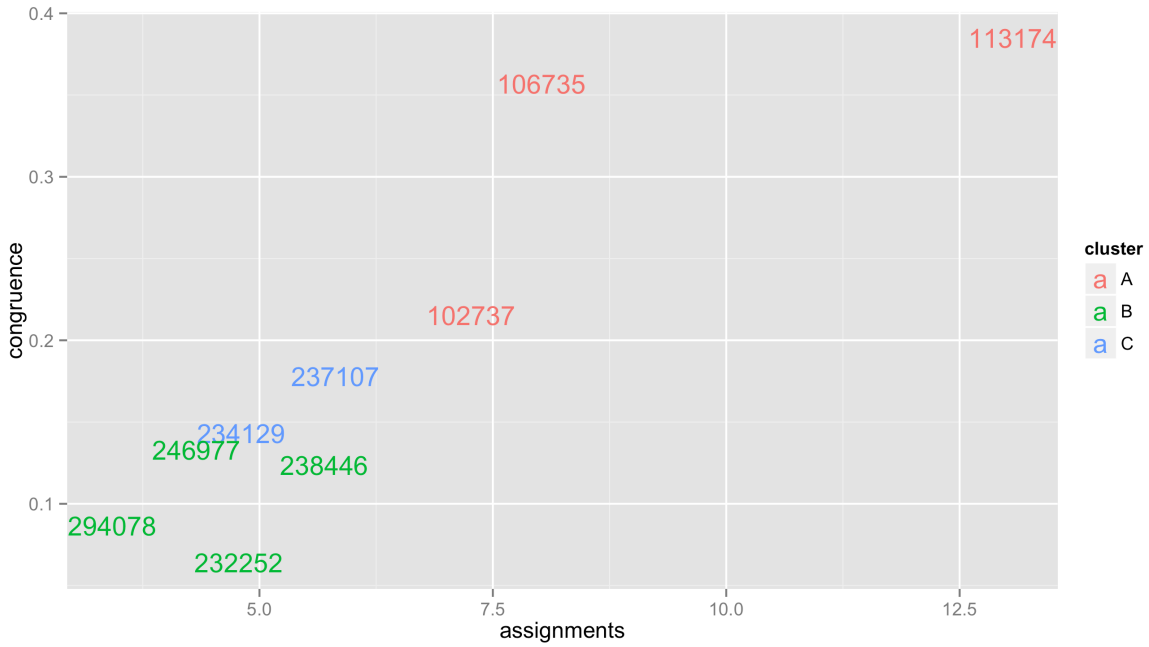

Figure 3. *Scatter plot of average number of assignments versus average congruence over sequences of 100 comments in selected networks. The colors correspond to the three main clusters identified derived with hierarchical clustering on the basis of commonalities among email-addresses of participants to the networks. The correlation coefficient of assignments and congruence of the networks displayed here is 0.89 ($p = 0.0011$).*

Acknowledgements

This research received financial support from the French National Research Agency through the programs "Investments for the Future" for LabEx Entreprendre and "CONTINT" for project OCKTOPUS.

References

Amrit, C., & van Hillegersberg, J. (2008), Detecting Coordination Problems in Collaborative Software Development Environments, *Information Systems Management* 25(1).

Bailey, S. E., Godbole, S. S., Knutson, C. D., & Krein, J. L. (2013, October). A Decade of Conway's Law: A Literature Review from 2003-2012. In *Replication in Empirical Software Engineering Research (RESER), 2013 3rd International Workshop on* (pp. 1-14). IEEE.

Bolici, F., Howison, J., & Crowston, K. (2009, May). Coordination without discussion? Socio-technical congruence and Stigmergy in Free and Open Source Software projects. In *Socio-Technical Congruence Workshop in conj Intl Conf on Software Engineering, Vancouver, Canada*.

Cataldo, M., & Herbsleb, J. D. (2013). Coordination Breakdowns and Their Impact on Development Productivity and Software Failures. *IEEE Transactions on Software Engineering*, 343-360.

Cataldo, M., Herbsleb, J. D., & Carley, K. M. (2008, October). Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement* (pp. 2-11). ACM.

Carley, K. M., & Svoboda, D. M. (1996). Modeling organizational adaptation as a simulated annealing process. *Sociological Methods & Research*, *25*(1), 138-168.

Colfer, L., & Baldwin, C. (2010). The mirroring hypothesis: Theory, evidence and exceptions. *Harvard Business School Finance Working Paper*, (10-058).

Conway, M. E. (1968). How do committees invent. *Datamation*, *14*(4), 28-31.

Coplien, J.O., (2006), Organizational Patterns: Beyond Technology to People, in I. Seruca et al. (eds.), *Enterprise Information Systems VI*, 43–52, Springer, The Netherlands.

Crowston, K. E., Howison, J., & Annabi, H. (2006), Information Systems Success in Free and Open Source Software Development: Theory and Measures. *Software Process Improvement and Practice*, 11, 123–148.

Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2013). Leveraging Transparency, *Software, IEEE, 37-43.*

Dalle, J. M., & David, P. A. (2005). The allocation of software development resources in 'open source'production mode in Joe Feller, Brian Fitzgerald, Scott Hissam, Karim Lakhani, eds., *Making Sense of the Bazaar*, MIT Press.

Dalle, J. M., David, P. A., Ghosh, R. A., & Wolak, F. A. (2004). Free & Open Source Software Developers and 'the Economy of Regard': Participation and Code-Signing in the Modules of the Linux Kernel. In *Oxford Workshop on" Libre Source 'convened at the Oxford Internet Institute* (pp. 25-26).

Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, *85*(6), 1213–1221. doi:10.1016/j.jss.2012.02.033

De Leede, J., & Looise, J. K. (2005). Innovation and HRM: towards an integrated framework. *Creativity and innovation management*, *14*(2), 108-117.

den Besten, M. L. & Dalle, J.-M. (2008). Keep it Simple: A Companion for Simple Wikipedia? *Industry & Innovation*, *15*(2):169-178.

den Besten, M. L., Dalle, J.-M. & Galia, F. (2008). The allocation of collaborative efforts in open-source software. *Information Economics and Policy*, *20*(4):316-322.

Feller, J., Finnegan, P., & Hayes, J. (2006). Open source networks: an exploration of business model and agility issues. *ECIS 2006 Proceedings*. Retrieved from http://aisel.aisnet.org/ecis2006/95

Gaudeul, A. (2007). Do open source developers respond to competition? The LaTeX case study. *Review of Network Economics*, *6*(2).

Glance, N., & Huberman, B. (1993). Organizational fluidity and sustainable cooperation. *arXiv preprint chao-dyn/9307015*.

Guo, P. J., Zimmermann, T., Nagappan, N., & Murphy, B. (2011, March). Not my bug! and other reasons for software bug report reassignments. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work* (pp. 395-404). ACM.

Halloran, T. J., & Scherlis, W. L. (2002, May). High quality and open source software practices. In *2nd Workshop on Open Source Software Engineering*.

Henderson, R. M., & Clark, K. B. (1990). Architectural innovation: the reconfiguration of existing product technologies and the failure of established firms. *Administrative science quarterly*, 9-30.

Johnson, J. N., & Dubois, P. F. (2003). Issue tracking. *Computing in Science & Engineering*, *5*(6), 71-77.

Klincewicz, K. (2006). Innovativeness of open source software projects.

Kwan, I., Cataldo, M., & Damian, D. (2012). Conway's Law Revisited: The Evidence for a Task-Based Perspective. *Software, IEEE*, *29*(1), 90-93.

Langlois, R. N. (2007). The Entrepreneurial Theory of the Firm and the Theory of the Entrepreneurial Firm. *Journal of Management Studies*, *44*(7), 1107-1124.

Langlois, R., & Garzarelli, G. (2008). Of Hackers and Hairdressers: Modularity and the Organizational Economics of Open-source Collaboration. *Industry & Innovation*, *15*(2), 125–143.

Levandowsky, M., & Winter, D. (1971). Distance between Sets. *Nature*, *234*(5323), 34–35. doi:10.1038/234034a0

MacCormack, A., Rusnak, J., & Baldwin, C. Y. (2006). Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Management Science*, *52*(7), 1015-1030.

MacCormack, A., Baldwin, C., & Rusnak, J. (2012). Exploring the duality between product and organizational architectures: A test of the "mirroring" hypothesis. *Research Policy*, *41*(8), 1309-1324.

Mateos Garcia, J. and Steinmueller, W. E. (2003). Applying the Open Source Development Model to Knowledge Work. INK Open Source Research Working Paper, SPRU, Brighton, UK.

Nordberg III, M. E. (2003). Managing code ownership. *Software, IEEE*, *20*(2), 26-33.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, *15*(12), 1053-1058.

Raymond, E. S. (2001). *The Cathedral and the Bazaar. Musings on Linux and Open Source by an Accidental Revolutionary*, revised edition. Sebastopol, CA: O'Reilly and Associates.

Sandusky, R. J., Gasser, L., & Ripoche, G. (2004). Bug Report Networks: Varieties, Strategies, and Impacts in a F/OSS Development Community. In *Proceedings of the 1st International Workshop on Mining Software Repositories (MSR 2004)*.

Sandusky, R. (2005, July). Software problem management as information management in a F/OSS development community. In *Proceedings of the First International Conference on Open Source Systems* (pp. 44-49).

Thom-Santelli, J., Cosley, D. R., & Gay, G. (2009, April). What's mine is mine: territoriality in collaborative authoring. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1481-1484). ACM.

Tsay, J., Dabbish, L., & Herbsleb, J. D. (2013). Social Media in Transparent Work Environments, *Collaborative and Human Aspects of Software Engineering* (pp. 65-72). San Francisco, CA.

Villa, L. (2003, July). Large free software projects and Bugzilla. In *Proceedings of the Linux Symposium* (pp. 23-26).

West, J., & O'Mahony, S. (2008). The role of participation architecture in growing sponsored open source communities. *Industry and Innovation*, *15*(2), 145-168.