



Dynamics of Institutions and Markets in Europe is a network of excellence of social scientists in Europe, working on the economic and social consequences of increasing globalization and the rise of the knowledge economy.
<http://www.dime-eu.org/>

DIME Working Papers on INTELLECTUAL PROPERTY RIGHTS



Sponsored by the
6th Framework Programme
of the European Union

<http://www.dime-eu.org/working-papers/wp14>

Emerging out of DIME Working Pack:

'The Rules, Norms and Standards on Knowledge Exchange'

Further information on the DIME IPR research and activities:

<http://www.dime-eu.org/wp14>

This working paper is submitted by:

Matthijs den Besten^{*}, Jean-Michel Dalle^{} and Fabrice Galia^{***}**

^{*}Oxford e-Research Centre

Email: matthijs.denbesten@oerc.ox.ac.uk

^{**}Université Pierre et Marie Curie (Paris VI)

Email: jean-michel.dalle@upmc.fr

^{***}Burgundy School of Business and ERMES-CNRS, University Panthéon-Assas (Paris II)

Email: galia@escdijon.com

The Allocation of Collaborative Efforts in Open-Source Software

**This is Working Paper
No 72 (May 2008)**

The Intellectual Property Rights (IPR) elements of the DIME Network currently focus on research in the area of patents, copyrights and related rights. DIME's IPR research is at the forefront as it addresses and debates current political and controversial IPR issues that affect businesses, nations and societies today. These issues challenge state of the art thinking and the existing analytical frameworks that dominate theoretical IPR literature in the fields of economics, management, politics, law and regulation- theory.

The Allocation of Collaborative Efforts in Open-Source Software¹

Matthijs den Besten¹, Jean-Michel Dalle^{2,*} and Fabrice Galia³

¹ Oxford e-Research Centre, 7 Keble Road, OX1 3QG, Oxford, UK

<matthijs.denbesten@oerc.ox.ac.uk>

² Université Pierre et Marie Curie (Paris VI), c/o Agoranov, 3 rue Castex,
75004 Paris, France

<jean-michel.dalle@upmc.fr>

³ Burgundy School of Business and ERMES-CNRS, University Panthéon-
Assas (Paris II), 21, rue Sambin, 21000 Dijon, France

<galia@escdijon.com>

* Corresponding author

Abstract. The article investigates the allocation of collaborative efforts among core developers (maintainers) of open source software by analyzing on-line development traces (logs) for a set of 10 large projects. We specifically investigate whether the division of labor within open-source projects is influenced by characteristics of software code. We suggest that the collaboration between maintainers tends to be influenced by file vintage, by

¹ This research has been partly supported by *Calibre*, an EU FP6 Coordination Action. The support of the US National Science Foundation (NSF Grant n° IIS-0326529 from the DTS Program) is also gratefully acknowledged by one of us (JMD), as is the e-Horizons grant from the James Martin School by another (MdB). A preliminary version of this work had been presented at the Second International IFIP Conference on Open-Source Systems (OSS2006) at Como in June 2006, where it won the ‘Best Paper’ award. The authors would also like to thank several colleagues for interesting discussions about the issues addressed in this article, and notably Paul A. David and Stefan Koch. Last, but not least, we would like to thank two anonymous referees for their insightful comments.

modularity, and by different measures of the complexity of the code. We interpret these findings as providing preliminary evidence that the organization of open-source software development would self-adapt to characteristics of the code base, in a stigmergic manner.

Keywords: L17 - Open Source Products and Markets; L86 - Information and Internet Services; Computer Software; L23 - Organization of Production; D71 - Social Choice; Clubs; Committees; Associations; D02 - Institutions: Design, Formation, and Operations

1 Introduction

The recent focus, in economics, management and other disciplines, on “online communities”, “virtual communities” and related notions, signals an increased and increasing interest among scholars vis-à-vis the (more or less) novel organizational forms that populate the globalized world and that largely rely upon software and Internet technologies. In essence, these issues are structurally related to the modern theory of the firm and of its boundaries, from Coase (1937) and Williamson (1975) onwards. It is now clear that there exists a ‘gray’ zone of non-market relations and proto-organizations that surrounds firms, exists in between them, and which is increasingly able to flourish and self-develop by using the Internet. Moreover, it is becoming clearer that commercial firms cannot neglect these mostly non-commercial organizations as they are important sources of innovation, value, and profit.

We believe that the examination of open source software projects has a great potential to inform, if not drive, this important area of research. This is not simply because of the successes of many large projects like the Linux operating system, the Apache web server and the Firefox browser, or because the activity of large open-source communities is fully and freely available for investigation on the Internet, in minutely detailed form, or because open-source communities produce goods that have already acquired a very significant economic value. It is because the

organization of open source software production has long attracted the attention of scholars, since Raymond's (1998) seminal essay "The cathedral and the bazaar".

Raymond likened the development of the Linux kernel to the hubbub of the bazaar, as opposed to more traditional cathedral-like organizations. However, while the notion of a bazaar suggests a more or less flat, market-like anarchy, subsequent and numerous studies of open source software development have uncovered strong hierarchical layers in its organization. A small core of developers maintains the code base, and this team is helped by a larger group of co-developers who contribute pieces of code and by an even large group of bug-submitters and feature-requesters (e.g. Crowston and Howison, 2005). The fact that the Linux kernel project had to appoint a group of middle managers (known as "Linus' lieutenants", Linus Torvalds being the "benevolent dictator": Aaltonen and Jokinen, 2007) reveals a visible hand of management and of leadership in the organization of open-source software development.

In this context, between bazaars and highly centralized hierarchies, we believe that scientific investigations are crucial to determine what characterizes the organizational forms that open-source and other communities create and adopt and to distinguish which processes are specific about these organizational forms. In this respect, we have argued in previous works in favor of a *stigmergic theory* (Dalle & David, 2005, 2007). The theory is that communities would cognitively and collectively react to some of the signs (*stigma*, in ancient Greek) that characterize the collective output of the community – the code base, in the case of open-source software – precisely since this output is, by definition, *open*, i.e. fully accessible to anyone. Openness combined with highly autonomous agents who allocate their own efforts produces an organization whose processes become reactive and self-adaptive as a consequence of the semantic signs reflecting the work of others within the community. The result is a coordination of actions. The fact that a population of agents is statistically reactive to such signs allows for the collective development of stigmergic behavior. Stigmergy, inspired by self-organization theory, suggests that studying aggregate outcomes – for instance the existence of a code *tree* that reflects the architecture and the evolution of the code – is a means for revealing the hierarchy

of preferences among developers. In that sense, stigmergy would be semantics turned collective, where the emergence of coordination is influenced by semantic artifacts.

As a consequence, we suggest that the organizational forms associated with online communities might be called *softs*: first of all because we believe that important research has to be conducted concerning the organizational forms that online communities create and adopt, and therefore that they deserve a distinctive name. Second, precisely in order to distinguish them from the more traditional *firms* and to insist on the importance of dedicated studies of productive organizations adopted by virtual communities *per se*. Finally, stigmergic theory suggests that *softs should be softer than firms* – in the sense of more reactive and more adaptable.

In order to provide further supportive evidence to this claim, this article focuses on softs through the lens of software complexity. Since the allocation of efforts is a crucial characteristic of problem-solving organizations, and following in the footsteps of Simon (1946), we investigate how efforts, within large open-source software communities, are allocated to the resolution of *complex* tasks. To do so, we build upon the fact that software complexity is a well-known concept in computer science, for which metrics have been developed, i.e., standardized ways to measure the complexity of a piece of software code. Simply put: we would like to know whether developers work differently on different pieces of code depending on their complexity, which would support our claim that signs affect softs stigmergically, and therefore affect aggregative collective outcomes – that coordination within the community is influenced and oriented by semantic characteristics of the work in progress.

We first summarize research that has recently addressed some of the organizational aspects of open-source software communities. Next, we describe the detailed data we have retrieved from a set of open source software development projects and introduce a few methodological caveats concerning their use. We then present our analysis and findings as to how complexity affects collaboration and the allocation of efforts among developers. We conclude by making several suggestions for future work.

2 The Organization of Open Source Software Production

Open source software is a type of software that has become increasingly prevalent over recent years. In contrast to closed source or ‘proprietary’ software, the human-readable source code of an open-source software program is distributed along with the program itself – with the rights to modify it and in most cases with the rights to redistribute it. The most famous example of open-source software is Linux, an operating system based on Unix and developed by a community initiated and headed by Linus Torvalds. Very early on, Microsoft, a dominant player in the market for operating systems, acknowledged the strength of Linux as stemming from ‘new development methodology’ (Valloppillil, 1998) – effectively a new way of *organizing* software production. By 2006, Linux had been adopted by about 30 million users, and major companies like Oracle had started providing professional support for Linux. In addition, there were an increasing number of small and medium sized companies, the most prominent of which is Red Hat (NASDAQ: RHAT) offering open-source-based solutions to their customers. In June 2006, another prominent example of open-source software, the web server Apache, had a market share of 60.64% against 30.67% for its immediate competitor (Netcraft, 2006), not to speak of the success of the Web browser Firefox or the open-sourcing of the well-known Java technology by Sun Microsystems. Based on this ‘new development methodology’, open-source software has now reached the mainstream of the software industry and is experiencing change in its industrial organization.

However, an important part of the early literature on these issues was mainly focused on the motivations of open-source developers and on the relations of these motivations with more traditional market-based incentive structures. Only more recently have new developments of this literature focused more specifically on the ability of open-source communities to organize in order to develop marketable products. Drawing a parallel with open science, it has notably been suggested that reputation mechanisms played a role within large open-source communities (Himanen, Torvalds and Castells, 2001; Benkler, 2002; and Dalle and David, 2005). Benkler (2006) has extended this vision and characterized open-source production as “peer production”, a major contributor to the “wealth of networks”. In a related vein,

but retaining a focus on the motivation of *users* to participate to open-source projects, von Hippel has suggested that open-source communities behave as “horizontal innovation networks”.

These theoretical advances, although relevant for explaining various aspects of the open-source phenomenon, have fallen short in accounting for the prevalence of hierarchies and related governance structures that have been widely reported by empirical studies. This is indeed surprising. Economists are familiar with the difference between real and formal authorities (Aghion and Tirole, 1997). The role of reputation with open-source communities easily translates into meritocratic authoritative structures based on skills, competence and deeds. These may not be formal in a contractarian sense, but they are often based on explicit status mechanisms. Furthermore, there is no logical reason why the volunteer nature of open-source participation would necessarily prevent the existence of various kinds of incentive and governance mechanisms that could influence the allocation of efforts.

Softs are not flat, market-like, anarchies. Even if we put aside more detailed description of the various roles that exist within open-source communities, such as Crowston & Howinson’s (2005) ‘onion’ model, there is now consensus in the recognition that there almost always exist a distinction between *core* developers and more *peripheral* contributors. The former, often called maintainers, constitute the subset of developers who have the technical rights and access codes to properly modify and maintain the code base, whereas more peripheral members have to submit their potential patches to core members. Depending on the project, there are tens or hundreds of maintainers, who collectively constitute the core of the productive organization associated with these communities.

That said, very little is known yet about how maintainers allocate their efforts and select and coordinate their tasks. What determines their largely autonomous decisions about their individual work agendas in the context of a given project? What are the consequences to the aggregate behavior and performance of the individuals’ allocation of their efforts? Is the division of labor in softs similar to the one in traditional firms? In this context, using software complexity metrics, we investigate in this article whether the allocation of efforts is influenced by the complexity of the

code – a proxy, among others, for stigmergic dependence of the organization upon the code: of the soft upon software.

3 Project Data Selection and Limitations

Development activities are to a very large extent traceable in the activity logs that maintainers leave behind in their virtual environments. Indeed, open source software projects typically feature mailing lists where developers discuss their work and non-developers submit requests or ask for help. In addition, there may be discussion forums and bug tracking tools. Not least, the source code itself is available and, when, as is often the case, a version control system is employed, all older versions of the source code can be examined so that the development process can be traced back to the start. Development communities use version control systems to keep track of everything that is contributed to the code base, identifying when entered and by whom. When conflicts arise due to a change, a version control system makes it possible to undo that change and revert to the source code as it was before the change was made. The developers who have the authorization – "privilege" – to commit such changes are the maintainers: they are not always the authors of the patches incorporated in their changes, but they will almost surely have at least reviewed it and made sure of its possible inclusion in the specific part of the code that it targets.

Mining these free and easily accessible data has already proven of a great utility to software engineering research. Koch and Schneider (2002) studied efforts, cooperation and coordination in the open-source project Gnome. Mockus, Fielding and Herbsleb (2002) studied the pace with which bugs were resolved based on information in mailing lists and software logs. Scacchi (2002) performed an in-depth ethnographic analysis of the implicit ways in which requirements are gathered in open source projects. More recently, and relevant for our present study, Capiluppi et al. (2005) focused on the evolution of the complexity of procedures within the code.

To create a database adapted to our investigations, we have selected a set of 10 large open-source projects. By large, we mean that all of these projects have a

substantial amount of code and number of contributors as well as a development history with logs that typically span a period of five to ten years. This set also constitutes a collection that is diverse with respect to product type as well as target audience. In alphabetical order, we have selected:

- a web server – *Apache*,
- a version control system – *CVS*,
- an instant messaging application – *Gaim*,
- a compiler – *GCC*,
- an interpreter for the PostScript language and for PDF – *Ghostscript*,
- a web browser – *Mozilla*,
- an operating system – *NetBSD*,
- a secure networking protocol – *OpenSSH*,
- a database – *PostgreSQL*,
- and a programming language – *Python*.

Several of these projects, most notably Mozilla and Apache, have received attention from software engineering researchers; others have received less attention. Our ultimate objective is to try to distinguish features that are generally associated with the open-source mode of software production, and to observe differences among projects which would be the consequences of their organizations and maintenance policies as they have been set up and designed as specificities of their softs.

The data that we looked at for our investigations was extracted from logs of development activity generated by software version control systems. For each project in the selection, we extracted CVS development logs. CVS is the most widely used version control system for open source software development and its logs are relatively easy to analyze using lexicographic scripts. We can identify, for each file in each project, each revision of that file and for each of these revisions, when the revision was made, who was responsible for the revision and how many lines of code were added to and deleted from the file as a result of the revision. At this level of analysis, we constrain our sample to all the files that contain source code written in C or C++ i.e. to files with *.c*, *.C*, *.cc*, or *.cpp* suffixes. Clearly, we thus ignore some of the files in which there is code, For example, there are likely significant omissions

in the Python project where one would expect that much of the code is written in the language developed by the project, namely Python. Similarly, there are probably files in some of the projects that should not be included in our analysis of source code files, for instance test files in GCC. This is a limitation that we expect to be minor considering the number of files in our analysis, but it deserves to be addressed in future studies.

For each of the 10 projects, we computed descriptive data reported in part in Table 1. More specifically, for the purpose of studying the allocation of efforts in softs, we computed for each file in the sample, and for each month in its history, how many distinct maintainers had committed a change during that month, and how many commits each file had received during that month.

Before presenting our investigations and their results, several caveats have to be mentioned:

1. The earliest record in the logs does not necessarily coincide with the start of the project itself, as the decision to adopt CVS could have been made later into the development of the project: A case in point is GCC, which started well before the first recorded commit in 1997.
2. It is not always perfectly clear where the boundaries of a given project are. For instance, Apache and Mozilla have their own repositories but both host multiple applications. Lacking a clear rule for now about where to draw these limits, we decided that in the case of Apache, we would restrict ourselves to the logs concerning Apache HTTP Server 2.0. In the case of Mozilla, we considered the whole suite. In the case of NetBSD, we only looked at the kernel of the operating system, while in the case of OpenSSH, which is part of OpenBSD, we focused at the subdirectory within OpenBSD where OpenSSH resides.
3. We only consider the main development branch and ignore activity within other branches.
4. It is not always clear when a file is really part of the project's code base. Most files are deleted when they are no longer needed, but we cannot be sure that this policy is always followed. Furthermore, a few files are “born

dead” (which can happen when a file is created in a branch other than the main branch), and sometimes files registered as dead are “revived”. These details are a consequence of the way that CVS is implemented and deployed.

5. Finally, and although this is expected to be negligible, it might be necessary to investigate in further research whether CVS accounts could be used by more than one maintainer, which could create a potential source of bias.

4 Analysis

The econometric tests presented here focus on the number of maintainers per month. This variable is considered as an indicator of collaboration and collaborative maintenance. In this context, previous investigations (den Besten and Dalle, 2005; den Besten, Dalle & Galia, 2006) had attracted our attention to the variability through time of collaborative maintenance and of activity on a given file. We had typically observed that in up to 80 or 90% of the cases, only one maintainer committed a change to a given file during a given month; and that activity tended to be concentrated during the first weeks in the life of a file. As a consequence, we investigate here the *maximum* number of maintainers per month over the period in which the file has existed (*max maint's*). We therefore focus our investigations on the actual periods of development, compared to periods of low activity that occur when a file is mostly “done”.

To address the issue of how softs self-allocate efforts in relation to the complexity of tasks, we tested various measures of complexity used by software engineering research, and we selected 2 different measures of the complexity of the problems addressed by a given piece of code in a file: the *Halstead volume* of the file (*Halstead*), and the maximum *McCabe complexity index* for all functions in the file (*McCabe*). Halstead volume is a measure that combines the length of a program – its total number of operators and operands² – together with its vocabulary – its total number of distinct operators and operands. McCabe’s cyclomatic complexity, by

² In $(a + b)$, '+' is the operator and 'a' and 'b' are the operands.

contrast, is based on the number of independent paths that could be taken through the code and is notably related to the mental difficulty of a programming task, since increasing the number of paths could make a program more difficult to understand (Curtis et al., 1979). The complexity of a file is therefore measured both "additively" – how many items are manipulated by the program: *Halstead* – and “algorithmically” – how many different paths: *McCabe*.

Finally, other variables used in the regressions are proxies for the size, age, and modularity of files: the size of a file is represented as its number of lines of code (*LOCs*), its age by its creation date (*Youth*), and its modularity by the number of functions it contains.

It is important to note that these variables are obviously not fully independent from one another, though they capture different characteristics of software code. This is the reason why, in addition to traditional regressions presented here, we also ran several tests using more advanced techniques. These included partial least square regressions (PLS) and principal component regressions (PCR). Both techniques are known to be more robust notably vis-à-vis the signs of the coefficients in the case of colinearity. These tests confirmed the signs obtained using linear regressions³.

We ran several tests for all 10 projects on our database, trying to explain *max maint*, our measure of collaborative activity, using the complexity and control variables. We also created two benchmarks whose aim is to more generally represent the “open-source software mode of production”. The first one (*ALL*) was created by merging the 10 databases associated with each project, while the second (*ALL WEIGHTED*) was created by merging all 10 databases, assigning a weight to each entry (file) of the form $1/N$ where N was the number of files without missing variables in the database of that particular project. Therefore, *ALL WEIGHTED* is less likely to over-represent the bigger projects, Table 2 presents detailed results for Apache and for *ALL WEIGHTED*, while Table 3 summarizes our results for all projects.

³ We present here only the results of linear regressions in a summarized form: more detailed linear results and results obtained with PLS and PCR techniques are available upon request from the authors.

5 Results

In examining the regressions described in the previous section, we first observe significant regularities across projects with respect to the influence of complexity variables and other variables on the allocation of collaborative work. More recent files tend to have lower values of *max maint* (lower instances of collaboration); on the contrary, larger files (in terms of LOCs) and more modular files (with more numerous functions) have higher values of *max maint*, i.e. more collaborative work. This latter finding could be consistent with the fact that modularity eases collaboration by reducing interdependences, while the effect of size probably corresponds to the intuitive idea that bigger files imply the collaboration of more numerous developers. The former finding could be consistent with the fact that the most important files are created early in the development of a project.

Second, *McCabe* complexity is significantly correlated with increased collaboration as measured by *max maint*, while *Halstead* complexity has the opposite effect. A possible interpretation for this finding might be related, on the one hand, with the attractiveness of complex problems, in the *McCabe* sense, to open-source developers, which would be consistent with the prevailing theories about reputation-based and skill-based motivations. On the other hand, the lack of attractiveness of complexity, in the sense of *Halstead* complexity, could indeed be discouraging to collaboration because of this measure's association with the abstruse nature of the code. In general, maintainers would rather work on more interesting and intellectually motivating code, an interpretation that is supported here by the significance and opposite signs of *McCabe* and *Halstead* measures.

Third, there are differences among the projects relative to the benchmark cases. These could be due to the particular technical nature of each project, and by consequence to the nature of the files that constitute it, or to different ways of composing files. We found some support for the former explanation in several cases: in the case of GCC, for which *Youth* is not significant but for which the code repository was ported to CVS years after the project was created; for OpenSSH, where the number of files is around 100 and therefore much more limited than for most other projects; and for Python, where the significance of *McCabe* could be lost

because we only focused on C and C++ files, whereas many files are written in Python.

Two projects, Ghostscript and Mozilla, however, show results that deviate from the two benchmark (aggregated) regressions. *LOCs* is significant and negative for Ghostscript, contrary to its sign in all the other projects and to both benchmarks, while *Halstead* is significant and positive for both Ghostscript and Mozilla, contrary again to all other projects and to both benchmarks. In this respect, it is striking to note that Ghostscript and Mozilla had the lowest and the highest number of maintainers, respectively, among all 10 projects studied. Furthermore, compared to others, both projects have a history with pronounced episodes of commercial involvement. Needless to say, this preliminary finding and its very preliminary and tentative interpretation will imply further investigations to determine whether there could exist an optimal range for the number of maintainers in a project, outside which the open-source development methodology would work differently and perhaps inefficiently.

Generally speaking, our investigations tend to suggest that several characteristics of software code, including its complexity measured with different metrics, are correlated with the allocation of collaborative efforts in the open-source software mode of production. That is to say, our findings and especially the correlation between complexity and the allocation of collaborative efforts provide support for the stigmergic theory of open-source software development. They also, indirectly, support the reputation-based, problem-solving, skill-based theory of the inner workings of open-source communities.

To state this conclusion differently, these results, and the stigmergic theory for which they provide empirical evidence are consistent with a *generalized Conway's Law*. According to Conway's Law (Conway, 1968), the architecture of the organization producing a given piece of software – thus a firm in general, or a soft in the case of communities – is largely congruent and isomorphic to the architecture of this piece of software. This empirical law has been largely verified and the subject of much debate in computer science and software engineering (e.g. Herbsleb & Grinter, 1999). The stigmergic theory of open-source development suggests that development organizations, when possible and thus notably when *soft enough*, are more generally

influenced by several characteristics of software, including complexity and modularity⁴, perhaps because complexity and modularity modify not only the granularity of efforts and the needs for coordination, but also, in the case of complexity, the attractiveness of tasks.

5 Conclusion

This article has documented investigations of detailed development records in order to study the allocation of collaborative efforts in open-source software. Based upon the fact that organizations – softs – set up by open-source software communities are neither bazaars nor cathedrals, we have presented preliminary evidence according to which the allocation of collaborative work was correlated with characteristics of the code, and notably to its complexity both in the sense of the Halstead and McCabe measures, but in opposite ways. We have suggested that these findings support the stigmergic theory of open-source development: softs self-organize in a way that is partly *governed* by characteristics of the code base. We have further suggested, although even more tentatively, that the existence of such regularities could be used to characterize the open-source mode of production, and therefore to identify deviations that could in some cases be related to organizational inefficiencies.

In this last respect, we believe that further investigations would be of particular significance at a time when numerous commercial entities are developing relationships with softs, typically by assigning salaried workers to open-source projects. To what extent do such practices modify the inner workings and the organizational processes of self-organized softs, perhaps transforming them into quasi-firms? Is this favorable, or does it hinder the ‘superior’ properties of the open-source development methodology? More simply, shouldn’t firms understand softs better, if they are to engage in a strategic partnership with them since their future profits will partly rely on the effectiveness of soft production? Furthermore, softs represent a type of organization that is more malleable than a firm, but maybe harder to manage and govern. The malleability of softs could, for instance, be reflected in

⁴ About modularity, a finding largely consistent with Mc Cormack et al. (2006).

the behaviour of “forking” where developers defect to a new project. Correspondingly, in softs, leaders and governance are perhaps more easily challenged, even if not abandoned (Dalle, 2003). On the other hand, it can be difficult for a group of volunteers to act strategically and to adjust their strategies to changing competitive circumstances (Gaudeul, 2007).

In this article, we have tried to move toward empirically enhancing our understanding of the open-source mode of production and of online communities. Further work is obviously required to address the limitations in our dataset. Even more important, it would be desirable to use a wider set of variables and more fine-grained econometric inquiries to pursue the questions addressed here. In addition, we believe that these works could also be supported by more *historical* investigations about open-source organizations – i.e. about how, when and why softs were set up as a consequence of the particular history of projects.

REFERENCES

- Aaltonen, T., Jokinen, J., 2007. Influence in the Linux Kernel community. In: Feller J., Fitzgerald B., Scacchi W., Sillitti A. (Eds.), *Open Source Development, Adoption and Innovation*. IFIP volume 234, Springer, New York, pp. 203-208.
- Aghion, P., Tirole, J., 1997. Formal and real authority in organizations. *Journal of Political Economy* 105, pp. 1-29.
- Benkler, Y., 2002. Coase's penguin, or Linux and the nature of the firm. *Yale Law Journal* 112, pp. 369-446.
- Benkler, Y., 2006. *The Wealth of Networks: How Social Production Transforms Markets and Freedom*. Yale University Press, New Haven.
- Brooks, F. P., 1995. *The Mythical Man Month*. Addison-Wesley Longman Publishing Co., Boston.
- Capiluppi, A., Faria, A. E., Ramil, J. F., 2005. Exploring the relationship between cumulative change and complexity in an open source system. *Proceedings of the Ninth European Conference on Software Maintenance and Reengineering*.
- Coase, R. H., 1937. The nature of the firm. *Economica* 4, pp. 386-405.

- Conway, M., 1968. How do committees invent? *Datamation* 14, pp. 28-31.
- Crowston, K., Howison, J., 2005. The social structure of free and open source software development. *First Monday* 10, http://www.firstmonday.org/issues/issue10_2/crowston/.
- Curtis, B., Sheppard, S. B., Milliman, P., Borst, M. A., Love, T., 1979. Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics. *IEEE Transactions on Software Engineering* SE-5, pp. 96-104.
- Dalle, J.-M., 2003. Open-Source technology transfers. Working Paper presented at the 2nd EPIP Workshop, MERIT, University of Maastricht, November 24-25th.
- Dalle, J.-M., David P. A., 2005. The allocation of software development resources in open source production mode, in: Feller, J., Fitzgerald, B., Hissam, S, and Lakhani, K. (Eds.), *Perspectives on Free and Open Source Software*. The MIT Press, Boston, pp. 297-328.
- Dalle, J.-M., David, P. A., 2007. Simulating code growth in Libre (open source) mode, In: Curien N., Brousseau, E. (Eds), *Internet and Digital Economics*, Cambridge University Press, Cambridge, pp. 391-422.
- den Besten, M. L., Dalle, J.-M. 2005. Assessing the impact of product complexity on organizational design in open source software. In: *Proceedings of the ECCS 2005 Satellite Workshop: Embracing Complexity in Design*.
- den Besten, M. L., Dalle, J.-M., Galia, F., 2006. Collaborative maintenance in large open-source projects. In Damiani E., Fitzgerald B., Scacchi W., Scotto M. (Eds.), *Open Source Systems*. IFIP volume 203, Springer, New York, pp. 233-244.
- Gaudeul, A., 2007. Do open source developers respond to competition? The (La)TeX case study. *Review of Network Economics* 6, pp. 239-263.
- Herbsleb, J. D., Grinter, R. E., 1999. Splitting the organization and integrating the code: Conway's law revisited. In: Boem B., Garlan D., Kramer J. (Eds.), *Proceedings of the 21st International Conference on Software Engineering*, pp. 85-95.
- Himanen, P., Torvalds, L., Castells, M., 2001. *The Hacker Ethic*. Secker & Warburg.
- Koch, S., Schneider, G., 2002. Effort, cooperation and coordination in an open source software project: Gnome. *Information Systems Journal* 12, pp.27-42.

- 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, pp. 1015-1030.
- Mockus, A., Fielding, R. T., Herbsleb, J. D., 2002. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology* 11, pp. 309–346.
- Netcraft, 2006. June 2006 Web Server Survey.
http://news.netcraft.com/archives/2006/06/04/june_2006_web_server_survey.html.
- Raymond, E. S., 1998. The cathedral and the bazaar. *First Monday* 3.
www.firstmonday.org/issues/issue3_3/raymond/
- Scacchi, W., 2002. Understanding the requirements for developing open source software systems. *IEE Proceedings – Software* 149, pp. 24–39.
- Simon, H. A., 1946. The proverbs of administration, *Public Administration Review* 6, pp. 53 – 67.
- Valloppillil, V., 1998. Open source software: A (new?) development methodology.
<http://www.catb.org/~esr/halloween/halloween1.html>.
- Von Hippel, E., 2007. Horizontal innovation networks – by and for users, *Industrial and Corporate Change* 16, pp. 293-315.
- Williamson, O., 1975. *Markets and Hierarchies*. Free Press, New York.

Table 1: Descriptive elements of the sample in the database.

	First month of act.	Files (#)	"c" files (#)	maint's (total #)	maint's (av)	max maint's (av)	revisions (av)	max revisions (av)
apache	07/96	4133	657	79	7.67	2.60	32.38	5.96
cvs	12/94	1062	287	30	3.67	1.41	23.74	3.01
gaim	03/00	5158	681	39	3.62	1.74	26.91	4.62
gcc	08/97	34757	16405	250	2.56	1.19	6.30	1.46
ghostscript	03/00	2819	932	23	3.68	1.76	9.08	1.76
mozilla	03/98	40545	8370	595	7.77	1.90	21.11	3.31
NetBSD	03/93	19514	7081	267	6.48	1.66	18.00	2.94
openssh	09/99	289	138	50	5.32	2.21	35.56	4.93
postgresql	07/96	4102	904	25	4.53	1.92	42.00	4.38
python	08/90	4643	419	88	5.94	1.94	31.59	4.78

Table 2: Estimations (OLS) for APACHE and ALL-WEIGHTED benchmark.

	APACHE	ALL WEIGHTED
	<i>Max maint's</i>	<i>Max maint's</i>
<i>Intercept</i>	2.53270*** 0.12740	1.71660*** 1.06E-2
<i>LOCs</i>	-1.25E-4 5.74E-4	6.80E-4*** 2.79E-5
<i>McCabe</i>	8.64E-3 ^x 4.12E-3	6.93E-2*** 3.31E-4
<i>Halstead</i>	1.25E-5 1.70E-5	-5.52E-6*** 6.68E-7
<i>Functions</i>	4.80E-2*** 1.11E-2	7.61E-3*** 5.33E-4
<i>Youth</i>	-2.91E-2*** 3.43E-3	-2.25E-3*** 5.87E-5
<i>Adj. R²</i>	0.4134	0.3354

Dependent variables: max. number of maintainers per month and max. number of revisions per month (parameter estimate, above, and standard error, below).

Confidence levels: *** < 0.0001 / ** < 0.001 / * < 0.01 / ^x < 0.05 / [§] < 0.10

Table 3: Econometric estimations (OLS) for all 10 projects.

	Adj. R2	LOCs	McCabe	Halstead	Functions	Youth
Apache	0.41		(+) ^x		(+) ^{***}	(-) ^{***}
CCVS	0.52	(+) ^{***}	(+) [*]	(-) ^{***}		(-) ^{***}
Gaim	0.46		(+) ^{**}		(+) ^x	(-) ^{***}
GCC	0.56	(+) ^{***}	(+) ^{***}	(-) ^{***}	(+) ^{***}	
Ghostscript	0.58	(-)^{***}		(+)^{***}		(-) ^{***}
Mozilla	0.28		(+) ^{***}	(+)[*]	(+) ^{***}	(-) ^{***}
NetBSD	0.30	(+) ^{***}	(+) ^{***}	(-) [*]		(-) ^{***}
OpenSSH	0.67	(+) [§]	(+) ^x			(-) ^{***}
PostgreSQL	0.37	(+) ^{***}	(+) ^x	(-) ^{***}		(-) ^{***}
Python	0.56	(+) ^{***}		(-) ^{***}		(-) ^{***}
ALL	0.38	(+) ^{***}	(+) ^{***}	(-) ^{***}	(+) ^{***}	(-) ^{***}
ALL WEIGHTED	0.34	(+) ^{***}	(+) ^{***}	(-) ^{***}	(+) ^{***}	(-) ^{***}

Dependent variables: maximum number of maintainers per month.

Between parenthesis, the sign of the coefficient.

Confidence levels: *** < 0.0001 / ** < 0.001 / * < 0.01 / ^x < 0.05 / [§] < 0.10