The Allocation of Collaborative Efforts in Large Open-Source Software Projects

Matthijs den Besten¹, Jean-Michel Dalle²,* and Fabrice Galia³

¹ Oxford e-Research Center
<matthijs.denbesten@oerc.ox.ac.uk>

² Université Pierre et Marie Curie, Paris, France
<jean-michel.dalle@upmc.fr>

³ ESC Dijon
<galia@escdijon.com>

*Corresponding author

Abstract. The paper investigates the allocation of collaborative efforts among maintainers of open-source software by analyzing the on-line logs of a set of 10 large projects. We specifically inquire whether the allocation of contributions is influenced by some of the characteristics of software code. We suggest that collaboration among maintainers in large open-source projects tends to be positively influenced by file vintage (older files attract more collaborative efforts), by modularity (measured by the number of functions per file) and by maximum cyclomatic complexity (McCabe measure) of all functions in a file. The volume of files (Halstead measure), which is related to the size of the implementation of their algorithms is also generally significant but negatively. These findings could be related to the project-level organizations that actually produce open-source software, which we suggest to denote as softs – as opposed to firms –, and whose processes appear as partly governed by some of the characteristics of the code base in a stigmergic way.

1 Introduction and Rationale

Communities in general and virtual communities in particular currently enjoy an increased interest from scholars in economics and other disciplines. For, in a growing number of situations indeed, among which open-source software is a prominent example, virtual communities have appeared to be successful in productive activities, i.e. in the production of various economic goods.
However, no clear understanding exists yet of the conditions under which these productive activities are actually organized, to put it differently, of the conditions under which efforts are allocated in open-source software and related communities—even less so of their potential efficiency. The relatively numerous considerations about the motivations and incentives of contributors to these communities have obviously reflected the initial puzzlement about the mere existence and development of these communities, and about their ability to globally attract contributions. However, and although this conundrum is not yet fully resolved, it should not mask the fact that, if the efficiency of these communities considered as productive organizations had not allowed them typically to become competitive vis-à-vis existing, more traditional productive organizations—firms—most of the questioning about motivations and incentives would probably have faded, if ever existed. And in the absence of a traditional managerial hand, it is indeed not at all obvious how community members do produce economic goods collaboratively, especially when they are located in various parts of the world and mostly interact on-line.

Do these communities for instance follow traditional patterns such as the division of labor? Indeed, it has been consistently reported by numerous observers that such patterns tended to appear, and even quasi-hierarchical forms as in the well-known Linux community. More generally, there are generally a division between core developers and more peripheral users bug reporters. The former are often called maintainers: i.e. the sub-set of contributors who have the rights—access codes—to modify the code base. But among these maintainers—tens or hundreds, depending on the projects—who constitute the core of the productive organization associated with these communities, very little is known of how they allocate their efforts, of how they select and coordinate their tasks.

Then, there are clearly also self-organization patterns in the collective behavior of those communities that influence their productive outputs: more specifically, open-source production is probably influenced by a particular class of self-organization phenomena designed as ‘stigmergic’ (Dalle & David, 2005, 2007): stigmergy basically captures the idea that an important part of the coordination in productive activities is done not through direct interactions between the individual contributors but mediated by the code itself, i.e. via the resulting productive output of the all past contributions. The original inspiration in this respect was taken from the construction of their nests by social insects, as it was found that social insects do not only interact directly but that their interaction and the coordination of their efforts depend also on the mediation that observations of the nest under construction itself represents.

That said, if the allocation of efforts is influenced by self-organized phenomena, these phenomena are themselves not isolated from all the management processes, incentive and governance mechanisms and the many other features of the peculiar organizations that virtual communities adopt to fulfill their productive functions. In particular, these different organizational structures are suitable to have important consequences vis-à-vis collaborative productive outputs: e.g., it has for instance been suggested that the implementation of an “early release” rule in the release management process could increase the global utility of the software produced (Raymond, 1998; Dalle & David, 2005); or else, the introduction of a middle management structure (‘Linus’ lieutenants’) is said to have been instrumental at one point in allowing the Linux project to scale further.
More, we indeed believe that there is an important and yet relatively neglected research question that deals with properly studying these organizational forms that those virtual communities create and adopt, including how the allocation of efforts is self-organized and/or managed in virtual communities. In this respect, and in others, we suggest denoting these organizational forms as softs, not only to distinguish them from more traditional firms, but also to insist on the importance of dedicated studies of productive organizations adopted by virtual communities per se. The broader term ‘community’ includes many aspects beyond the economic and managerial ones; by focusing simply on economics and management, the question more specifically becomes whether virtual communities might have invented a relatively novel organizational form whose economic and managerial significance, if any, is yet unclear, but whose significance will not be assessed if an explicit and direct attention is not devoted at its study: an attention that we hope to contribute partially to by naming names, that is, by giving these productive organization a proper name.

But there are also more theoretical reasons for suggesting this new terminology. What characterizes softs, then? To start with, there is no specific legal apparatus applicable to softs as is the case for for-profit and non-profit entities in all countries. Then, there are obviously nothing like shareholders in softs: under no circumstances can a soft be formally claimed as the property of any individual or entity. That doesn’t mean that there can’t be written “statutes”, such as they exist for instance in the case of the Debian community: but the enforceability of the rules adhered to by the community, either written or not, would be largely unclear in front of ‘regular’ courts of justice. And it isn’t clear whether these pronounced differences in the legal contexts of softs and firms do translate into organizational differences, and if so, in what extent and on which particular aspects. Simply, within softs, rules and organizational processes remain mostly conventional if not informal – which is not to say that they do not play a role in the governance structure of the soft. In particular, nothing prevents the existence and impact of various organizational features including incentive and governance mechanisms, etc., relying for instance on real instead of formal authorities (Aghion & Tirole, 1997). Still, a relevant issue is whether mechanisms applicable in other contexts are applicable to softs, in a different legal context, and/or whether softs could conversely open the possibility of alternative novel mechanisms.

In this respect, a major defining characteristic of softs, compared to firms, is that they are composed of agents i.e. who decide autonomously and voluntarily of their work agendas, i.e. who decide about which tasks they will select for themselves – to which they will allocate their voluntary contributions and efforts without any formal or contractual obligation in this respect. To put it differently, softs are partly informal productive organizations composed of autonomous agenda-setting agents. Definitely, this characteristic does not prevent the existence of various kinds of incentive and governance mechanisms that would influence the allocation of efforts between autonomous agents. In particular, since most of the work undertaken by virtual communities is generally made public on the Internet – and therefore, if not known to all, at least knowable to anyone who cares to know –, softs can partly rely on

---

1 In this respect, the notion of ‘disposable’ or ‘challengeable’ leaders had for instance been suggested as a potentially original governance mechanism for open-source software communities (Dalle, 2003).
reputation mechanisms which tend to regulate the allocation of efforts and which, again, influence their governance structures. Again, if not formal, there often exist ‘meritocratic’ authoritative structures in softs, based on skills, competence and deeds, and sometimes on explicit status mechanisms.

However, a consequence of softs being defined as the productive organizations set up and adhered to by communities of autonomous agenda-setting agents is that the processes implemented by these organizations should, in a sense by definition, be softer and easier to change and evolve than the processes of more traditional organizations such as firms, since all agents can at any moment decide to stop their contributions to the soft and/or to switch to different aspect of its productive elements, and to cease to follow this or that process. Although not anything is possible at any moment to all agents, the degree of freedom that agents have in deciding of their tasks – both the level of their effort and, more profoundly perhaps, the very nature of their effort – is of unprecedented level compared to traditional firms.

This is not to say that they would be bazaars, as opposed to cathedrals: and indeed, another consequence of focusing the attention on softs as productive organizations associated with virtual communities is to take stock of couple of relatively significant misconceptions that have been persistent in the early literature on these issues. If they are not cathedral-like organizations contrary to many software firms either, the productive organization of virtual and specially of open-source communities are far from ‘bazaars’ – more or less, flat market-like anarchies –, contrary to what was initially suggested (Raymond, 1998). Furthermore, softs are not adhocracies nor organized anarchies, although their bear some resemblance with both of these organizational forms: for, they are considerable complex and richer in their possible organizational settings, including the existence of quasi-formal hierarchies and the ability of their organizational features to persist with time, compared to more transient phenomena.

By analyzing softs are a special sub-class of the many hybrid organizational forms associated with virtual communities and autonomous agents, specifically focused toward productive activities, economists and management scientists might be in a better position to address a number of important issues. For instance, firms that sell complementary products to the productions of softs need to understand the characteristics of these ‘soft products’ to be able to adapt their own ‘hard products’ efficiently and to optimize the complementarities and cost reductions between both. Furthermore, when such firms invest into ‘soft production’ by typically contributing work to a soft directly or by sponsoring a more formal organization associated with a soft – like a foundation or non-profit organization – they would need to understand better the consequences of their actions, in terms of governance, of potential benefits as well as of potential threats. Only with an appropriate understanding of the economics and managerial characteristics of softs, and specially of how work is allocated within softs to produce goods, will they be able to reach further optimization levels in designing their own business models.

Note that an important caveat has to do here with the implication of commercial developers, i.e. of salaried or paid employees of business firms in virtual communities and in softs, which is typically the case for many of the large open-source projects: up to what fraction of the maintainer and developer population are softs still softs, or when do they become quasi-firms? There is an issue here that is not simple, and that further investigations should absolutely address, all the more so as it probably depends on the nature of the contracts and of the commitments of salaried maintainers: are they committed to follow directives from the firm that pays them, or do they simply receive a subsidy to follow up on the work that they had initiated as voluntary community members and soft workers, without any further obligation concerning the nature of the work they do and therefore on the allocation of their work? This is clearly an issue of utmost importance.

There is also another misconception that softs might contribute to clarify. The fact that most members of virtual communities are users of the products that these communities, organized in softs, create, is of absolute importance specially if one wants to understand how these products are designed for instance. However, it doesn’t mean that sufficient light shouldn’t be shed on the various organizational forms adopted by softs, and on their consequence on the productive capacities and capabilities of communities. The user-driven nature of these communities is not only a necessary condition of their formation and operation, which is a novel and crucial point, but the attractiveness of these virtual communities is also linked to their success and audience, and this success is in turn linked to how well they organize themselves into softs and on how efficient they are in this respect. Hence, the capacity of virtual communities to attract contributors, which is key to their ultimate development and success, is partly dependent upon their organization – from the way they recruit and select new developers and maintainers, which has already been preliminarily documented elsewhere, to the way they orient, or not, the allocation of efforts among autonomous contributors and tend, or not, to try to align their motivations with collective goals.

Indeed, in all respects – vis-à-vis bazaar and user theories, but also because of the impact of softs on associated commercial production –, we believe that a proper analysis of the organizational aspects of virtual communities, from incentives to signaling mechanisms via governance structures and the allocation of efforts is necessary. There is no magic at hand in virtual communities and in the softs that underlie their productive capacities, as it could seem when these aspects are misunderstood or considered negligible. On the contrary, the theory of soft production suggests that the success of a virtual community depends upon the organization it adopts, and on the economic and managerial processes it implements.

3 Again, analyzed through the lenses of softs, the widely acknowledged management and leadership skills of Linus Torvalds then refer to the almost unavoidable relevance of management and leadership in any human organization. Similarly, the decision of Torvalds to switch, at one point, to the more distributed BitKeeper code maintenance system, from the essentially centralized CVS technology, can be analyzed as implementing an improved information system, better adapted to the soft associated to the Linux community, in order also to support the growth of the organization beyond its previous limitations, and namely to implement in this case a organizational feature close to what is well-known as middle management.
Which isn’t a surprise at all, although the phenomena that characterize softs, beyond various qualitative observations, are yet largely unknown: however, they are relatively easily observable to economists and management scientists. Indeed, the activities of virtual communities and softs are in a considerable part traceable in the activity logs that maintainers and developers leave behind in their virtual environments. Open source software projects are natural candidates in this respect for quantitative empirical studies of softs, given their increasing economic success and the free and easy access they typically provide to such data. Preliminary steps in this direction have already been made in other contexts with other issues in mind: see e.g. Koch & Schneider (2002), Scacchi (2002), Robles et al. (2004)

In this context, we would like to focus here on an issue which we believe is of particular relevance for softs taken as organizations because it deals with complexity: more precisely, we would like to inquire about the nature of the organizational processes adopted by open-source softs to address the complex technical problems that arise during software development. The way efforts are allocated to the resolution of complex tasks is indeed, among various others, a very important and widely acknowledged characteristic of all organizations, specially when analyzed as problem-solving entities à la Simon. That is to say, we do not focus here on how softs solve complex problems, but on how they solve the problem of addressing complex problems. Furthermore, another reason to do this stems from the fact that complexity is a well-known characteristic of software goods (Brooks) and else because there are a few well-know and relatively standardized metrics for software complexity. Therefore, empirical investigations can rely in this respect on an existing body of knowledge from software engineering, which provides a relatively direct way of studying quantitatively a high-level organizational characteristic of softs.

Section 2 introduces open source software and reviews some of the relevant research done in that area. In section 3, we describe the database we studied and how we created it, and introduce a few important methodological caveats. In section 4, we follow by presenting findings as a result of our investigations. We briefly conclude by pointing out the limitations of the present study.

2 Open Source Software Production

Open source software is a type of software that has become increasingly prevalent over recent years and that, in contrast to closed source or ‘proprietary’ software, the human-readable source code of the 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 developed by a community initiated and headed by Linus Torvalds. Microsoft, a dominant player in the market for operating systems, acknowledged the strength of Linux early on (Valloppillil, 1998). By 2006, Linux had been adopted by about 30 millions of users, and major companies like Oracle had started providing professional support on Linux in addition to an increasing number of small and medium sized companies, the most prominent of which is Red Hat (NASDAQ: RHAT) that offer open-source-based solutions to their customers. Furthermore, in December 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. Meanwhile, the market share of the Web browser Firefox is surging despite the pre-existing dominance of Internet Explorer, and Sun has “open-sourced” the well-known Java technology. About 10 years after its apparition in its present form, open-source software has reached the mainstream of the software industry, and is rapidly modifying elements of its industrial organization.

In this context, it is not surprising that the attraction of numerous economists and management scientists has been attracted towards open-source software and related systems. An important part of the early literature on these issues has focused on the motivations of open-source developers, and on the relations of these motivations with more traditional market-based incentive structures. More recently, new developments of this literature have stressed the fact that another, important and puzzling issue had to do with the ability of open-source communities to organize themselves in a way that ultimately gives birth to marketable products. In this respect, the role of self-organization phenomena belonging to the stigmergic class has been highlighted as a potential theoretical explanation for some of the inner workings of open-source communities, so as to explain part of the coordination that exists between the more or less decentralized efforts within communities of distributed developers. In this respect, coordination stems also from interactions between developers in that they are mediated by the source code of the program, which is, by definition, quasi-perfectly accessible to all.

These findings, together with many others, have stressed even further the importance of all the governance structures, rules, incentive mechanisms, procedures, processes, routines, etc., that influence, orient and eventually determine the allocation of efforts within these communities. Their importance has been enlightened and stressed for most the large open-source projects by numerous pieces of work, though mainly anecdotal and mostly based on case studies, which had rendered difficult to progress toward generalizations as the conclusions were relatively heterogeneous, each being largely dedicated to particular projects. In particular, way beyond the largely misleading “bazaar” (Raymond, 1998), several case studies of open source software projects have shown that in many projects hierarchies tended to exist but that there was larger diversity in organizational forms from one project to the other than would have been expected. Indeed, peculiarities in organizational structures, from one open-source project to another, could unfortunately result in somewhat mistaking the trees for the forest, or conversely in not seeing the forest but only a mere collection of trees. Furthermore, this diversity in organizational structures did not allow for a proper analysis of which organizational features could be general and structurally characteristic of the open-source mode of producing software, and which were idiosyncratic, being implemented by this or that particular community.

For, the fact that interactions between developers are mediated by the existence of the source code and more specifically by some of its characteristics, such as its hierarchical architecture, does not preclude the fact that the individual choices of the developers are influenced not only by motivations and preferences, which when statistically significant at the level of the community modify the global aggregated outcome, but also by the characteristics of the organizations set up by development communities – by softs, as we have chosen to denote them. Among these
characteristics, reputation-based and status systems have received preliminary theoretical and empirical support and have been shown to be compatible with market-based incentives systems. Recent research has also pointed towards a so-called “onion model” of organization in which a core team of just a few developers is aided by a larger group of co-developers who are in turn aided by an even larger group of bug-submitters and feature-requesters, etc. (Crowston & Howison, 2005). In a similar vein, an “early release” rule could help create more numerous embryonic upstream modules within the code architecture, some of which would be developed by further contributions since their embryonic nature leaves room for further reputational rewards: this organizational rule or routine could be interpreted as giving birth to cooperative sub-games within the project, whose impact on the global utility was not zero-sum as it tended to attract more efforts toward upstream tasks and modules (Dalle & David, 2005).

More generally, the allocation of efforts within open-source communities reflects the autonomous decisions of developers about their individual work agendas in the context of a given project. It is therefore oriented and influenced by in interplay between some of the characteristics of the code and the productive organization adopted by the community – by its soft –. As it has been argued before, we believe that this could open a fruitful line of research, to which this article tries to contribute.

Furthermore, another element that makes open source software projects attractive as a topic for research is that virtually the whole development process is recorded and that the archives of these recordings are freely available for investigation. More in particular, 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. Last, but not least, the source code is available and, when, as is often the case, a version control system is employed, in fact all old versions of the source code so that the development process can be traced back to the start. Researchers of software engineering have started to make use of this wealth of data to inform their investigations. Notable examples are Scacchi (2002), who performed an in-depth ethnographical analysis of the implicit ways in which requirements are gathered in open source projects, and that of Mockus and Herbsleb (2002), who studied the pace with which bugs were resolved based on information in mailing lists and software logs. Hashler and Koch (2005) propose a larger scale mining of the available information and discuss what kind of questions could be explored on the basis of that information, while Capiluppi et al. (2005) focus on the evolution of complexity of functions.

Using such datasets, it is feasible to address issues that more directly aim at shedding some light on softs i.e. on the productive organizations associated with open-source software communities. Our particular investigation here has to do with how developers allocate their efforts collaboratively within projects, and whether this allocation is influenced by characteristics of software itself. We specifically focus here on how softs handle complex tasks, namely, the complex technical problems that arise during software development because of the algorithmic nature of software code. We wonder how the problem of how complex problems are addressed, is itself addressed within softs, and whether complex pieces of code tend to receive collaborative work, or not, so as to provide preliminary insights into a relatively high-level organizational characteristic of softs, and thus on softs themselves.
3 Empirical Database & Caveats

In this context, the data that we looked at for our investigations was extracted from logs of development activity generated by software version control systems. Version control systems are used by development communities in order to keep track of everything that is contributed to the code base, when and by whom: when conflicts arise due to a change, a version control system for instance 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 digital rights to commit the changes, and who are not necessarily the authors of the code incorporated in that change, but which they will have typically reviewed, are called the maintainers. Specific rules apply as to who is allowed to commit what, but there are always maintainers i.e. a limited subset of individuals who have digital rights in the version control system. They usually belong to the first and second key layers in the onion model, being either project leader or “just” maintainers. We specifically inquire in this article about how they are organized and allocate their efforts.

To create a database adapted to our investigations, we have selected a set of 10 open-source projects, attempting to create a collection that was diverse enough in terms of product complexity and targeted audience. In addition, the projects needed to be large i.e. to have a sufficient amount of code, contributors and development history: in the list below, the logs typically span a period of five to ten years. Furthermore, only those projects that provided easy access to their code repositories obviously could qualify. We thus suggest the following benchmark for large open-source projects (in alphabetical order):

- 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 already received attention from software engineering researchers. Others, like Gaim, stand out because of the amount or because of the length of their activity. Finally, and although we selected only large projects, we selected projects whose sizes were different (in terms of number of contributors, of files, and of years of history), as our ultimate objective is to try to discriminate between features generally associated with the open-source mode of software production within softs, and between differences among these projects which are consequences of their organizations and maintenance policies as they have been set up and designed by the softs.
Concretely, we extracted CVS development logs for all these projects. CVS is the most widely used version control system for open source software development and its logs are relatively easy to parse (e.g. Robles et al., 2005). These logs list, 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 (see the example presented in Annex). At this level of analysis, we restrain 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 miss in some of the projects at least, and probably more consistently in projects like Python, since they precisely deal with developing a programming language, some of the files in which there is code. 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 believe is minor considering the number of files in our analysis, but that we will lift in future studies.

For each of the 10 projects, we computed descriptive data reported partially in Table 1. More specifically for the purpose of studying collaborative maintenance as implemented by softs, we computed for each file in the sample and for each month in its history how many distinct maintainers had committed a change to that file during that month, and how many commits it had received.

Before presenting our investigations and their results, several further caveats have still 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 completely 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. For now, we only consider the main development branch and ignore activity in 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”. NB: All of this is mainly CVS-specific.

5. Finally, an although this is probably to be mostly ruled out, it might be necessary to investigate at some point whether CVS accounts could be used by more than one maintainer, which could create a potential source of bias.
4 Empirical Investigations

In order to study open-source software production, the econometric tests presented in this paper focus on two different measures for each file: the number of maintainers per month and the number of revisions per month. The first measure is considered as an indicator of collaborative maintenance, while the second addresses maintenance activity more generally. In this context, previous investigations (den Besten & Dalle, 2005; den Besten et al., 2006) have attracted our attention to the time variability of collaborative maintenance and activity on a given file: we have typically observed that in 80 to 90% of the cases, only one maintainer commits a change to a given file during a given month; and furthermore, that activity often tends to be concentrated during the first weeks in the life of a file. Therefore, we mainly investigate two dependent variables: the maximum number of maintainers per month over the period in which the file has existed (“max maint’s”) and the maximum number of revisions of files per month over this period (“max revisions”). These two variables thus focus on actual periods of intense development activity, compared to periods of low activity that occur when a file is mostly “done”.

Focusing on how softs self-allocate efforts in relation to the complexity of tasks, and after preliminary testing of various measures of complexity inspired by software engineering research, we selected 2 distinct measures of the complexity of the problems addressed by the code in a given file: namely, the Halstead volume of the file (“Halstead”), and the maximum McCabe complexity index for all functions in the file (“McCabe”). The complexity of a file is indeed at least both additive (how many items are manipulated by the program – Halstead) and cyclomatic (closer to how intrinsically complex a problem is – McCabe): in our estimations, Halstead represents the total Halstead volume of a file, while McCabe corresponds to the maximum McCabe complexity of all functions in a file. 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 inner modularity by its number of functions (“funs”).

At this point, 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 suitable in cases where colinearities are important: namely, partial least square regressions (PLS) and principal component regressions (PCR), which are known to be more robust notably vis-à-vis the signs of the coefficients. However, these tests confirmed the signs obtained using linear regressions. And the latter allow us also to address the issue of significance more directly, we present here only the results of linear regressions, and in a summarized form: more detailed linear results and results obtained through PLS and PCR techniques are available upon request from the authors.

We ran several tests for all 10 projects on our database, trying to explain these 2 dependent variables (max maint’s and max revisions) over these variables. We also created two different benchmarks whose aim is to more generally represent the “open-source software mode of production”, if any. The first one (“ALL”) was created by simply merging the 10 databases associated with each project, while the second, probably a more representative benchmark since it does not overweight the
bigger projects (ALL WEIGHTED), was composed in a similar way, by merging all 10 databases while associating 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. Table 2 presents detailed results for Apache and for ALL WEIGHTED, while Table 3 summarizes all results for Max Maint’s for all projects.

We observe that:

a. There are relatively large regularities across projects with respect to the influence of complexity and of other variables on collaborative work:

• Younger files tend to be less concerned by episodes of collaborative work; on the contrary, bigger files (LOCs) and more modular files (with more numerous functions) deal with episodes of more collaborative work. The latter could be consistent with the fact that modularity more or less by definition eases collaboration by reducing interdependences, while the effect of size probably corresponds to the intuition of “big” files implying the collaboration of more numerous developers. Finally, we suggest that most “interesting” and/or “relevant” files might be created earlier on during the development of a project: all the more so as other tests on the average number of maintainers per file indeed suggested that it was increased, controlling for file lifetime, for younger files (which could be simply related to the increase of the number of maintainers with time).

• About complexity, McCabe significantly induces more collaboration while Halstead has an inverse effect. These two dimensions of complexity represent therefore relatively opposed influences on collaborative work. A possible interpretation for this finding might deal with the attractivity of complex problems (McCabe) for maintainers, either because of reputational effects or because of the intrinsic complexity of solving these problems that would require efforts from more numerous developers; on the one hand, and with a relatively lower attractivity of files that would be complex in the sense of Halstead Volume, i.e. associated with a large total number of operators and operands – files in which the implementation of algorithms would have been of a larger ‘size’/complexity. This finding would therefore be consistent with the ideas that more maintainers would work on more interesting – intellectually motivating – code vis-à-vis less interesting, in a sense more verbose and/or less efficiently implemented, code.

• The comparison of Max Maint’s and Max Revisions regressions, specially when Max Maint’s is considered as an explanatory variable for Max Revisions, would tend to suggest that modularity (funs) and even more so youth and McCabe influence collaborative work directly, and collaborative activity only consequently. This provides further support to previous interpretations regarding the relative attractivity of the initial phases of open-source projects, and regarding the influence of modularity and of complexity in the sense of McCabe.

b. There are also limited differences among projects vis-à-vis this “benchmark” case, that could be due to either the particular technical nature of each project, and thus of the nature of the population of files that constitute it, or else to different rules also regarding how files are composed. We found some support for the former explanation in several cases: about GCC, for which Youth is not significant and for which the code repository was ported to CVS years after the project was created, which largely distorts the Youth variable; for OpenSSH, where the number of files is
around 100 and therefore significantly more limited than for most other projects: for Python, where the significance of McCabe for instance might be lost because we only focused on C and C++ files whereas many algorithms were implemented in Python. It is left to future, deeper, studies to determine whether some of these finer grained differences could be related to difference in organizational processes i.e. in the organization of related softs.

c. However, in this last respect, two projects present significant deviations from the benchmark case: Ghostscript and Mozilla. LOCs is significant but negative for Ghostscript, contrary to all other projects and to both benchmarks, and Halstead is significant and positive for both Ghostscript and Mozilla, compared again to all other projects and to both benchmarks. It is then relatively striking to notice that these two projects correspond to the lower and to the higher number of maintainers, respectively, among all 10 projects studied here. Needless to say, this finding deserves further investigations but it suggests that there could exist an optimal “range” for the number of maintainers in a given project and that it could result in sub-optimal episodes of collaborative work associated with a less efficient implementations of algorithms (Halstead); and at one end of this spectrum, developers could even not enough to work collaboratively on bigger files, if not discouraged. It would be consistent with and inefficient organization of associated softs that would have either ‘hired’ too many maintainers, or on the contrary would have remained too close and would face Malthusian consequences.

Generally speaking, and awaiting confirmation of these results on a larger collection of projects, and more detailed studies, our investigations tend to suggest that some of the characteristics of software code, including its complexity measured along different dimensions, would be related with the allocation of collaborative efforts in the open-source software mode of production. That is to say, they would provide some further support to the stigmergic theory of open-source software development: some of the characteristics of software influence the self-allocation of efforts among developers within softs. More specifically, our results would also provide further support to the reputational problem-solving theory about motivations that was suggested by several authors in academia in the open-source community.9,19

To push this line further, we would like to suggest that these results are consistent with a generalized version of Conway’s Law: according to Conway’s Law, the architecture of the organization producing a given piece of software – its soft, in the case of productive communities – influences when it does not replicate the architecture of this particular piece of software. This empirical law has been the subject of much debate in computer science and specially in software engineering, and still tends to be widely accepted. The stigmergy theory of open-source development supports the idea that the organization of softs would be more generally influenced by various other characteristics of the code base: the allocation of collaborative work is typically influenced not only by the size and modularity of files, and also by their complexity, yet in opposite directions depending on whether their complexity is intellectually attractive or not. It might be possible to identify other similar phenomena, and to show that the emergent processes that characterize softs are stigmergically influenced by the code base, and thus to extend Conway’s Law further.
5 Conclusion & Further Work

This article has documented investigations of detailed development records in order to study the allocation of collaborative efforts in large open-source projects. The success that many open-source projects have had in recent years and the voluntary nature of their development process provide a rationale for such investigations, together with the development of sponsorship by commercial entities that is on its part looking for more detailed approaches to the quality of code produced by different communities. Furthermore, we suggested that the softs installed by open-source software communities to organize their productive activities were neither bazaars nor cathedrals, and that these organizations that directly support production in open-source mode deserved more detailed studies per se.

More specifically, our contribution in this article has aimed at analyzing whether the allocation of collaborative work, taken as a particular and maybe a special process of softs, was related to some of the characteristics of the code, and specifically to its complexity both in the sense of Halstead (algorithm implementation) and in the sense of McCabe (cyclomatic complexity). We came to suggest conclusions that tend to support the stigmergic theory of open-source development: softs, which partly emerge in a self-organized way, are influenced and in a sense partly governed by characteristics of the code base. They seem to exhibit regularities that could be observed and studied, not only to characterize the open-source mode of production, but also possibly to identify deviations that could, in some cases at least, be related to organizational inefficiencies. If confirmed, these findings would be an significant step toward empirically enhancing our understanding of the open-source mode of production, and more generally of the productive activities associated with on-line communities.

However, we believe that cautiousness should absolutely prevail at this point: the main point in this article is perhaps simply that empirical investigations of open-source software development represent a fruitful area of research for economics and management science, and that they deserve further studies in many respects, including along the lines and suggested conclusions of this article – whether they will receive further support or on the contrary will deserve reassessments.

Acknowledgments

This research has been partly supported by Calibre, a EU FP6 Coordination Action. The support of the US National Science Foundation (NSF Grant No. IIS-0326529 from the DTS Program) is also gratefully acknowledged by one of us (JMD). 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 various colleagues for interesting discussions about the issues addressed in this article, and notably Paul A. David and Stefan Koch.
REFERENCES


Dalle, Jean-Michel (2003), Open Source Technology Transfers, presented at the 2nd EPIP Meeting, MERIT, University of Maastricht, November 24-25th.


Hahsler, Markus, and Stefan Koch (2005), Discussion of a Large-Scale Open Source Data Collection Methodology, Proc. HICSS 38.


ANNEX

Table 1: Descriptive elements of the sample in the database. Other statistics available upon request.

<table>
<thead>
<tr>
<th>Software</th>
<th>First month of act.</th>
<th>Files (#)</th>
<th>&quot;c&quot; files (total #)</th>
<th>maint's (av)</th>
<th>max maint's (av)</th>
<th>revisions (av)</th>
<th>max revisions (av)</th>
</tr>
</thead>
<tbody>
<tr>
<td>apache</td>
<td>07/96</td>
<td>4133</td>
<td>657</td>
<td>79</td>
<td>7.67</td>
<td>2.60</td>
<td>32.38</td>
</tr>
<tr>
<td>cvs</td>
<td>12/94</td>
<td>1062</td>
<td>287</td>
<td>30</td>
<td>3.67</td>
<td>1.41</td>
<td>23.74</td>
</tr>
<tr>
<td>gaim</td>
<td>03/00</td>
<td>5158</td>
<td>681</td>
<td>39</td>
<td>3.62</td>
<td>1.74</td>
<td>26.91</td>
</tr>
<tr>
<td>gcc</td>
<td>08/97</td>
<td>34757</td>
<td>16405</td>
<td>250</td>
<td>2.56</td>
<td>1.19</td>
<td>6.30</td>
</tr>
<tr>
<td>goscript</td>
<td>03/00</td>
<td>2819</td>
<td>932</td>
<td>23</td>
<td>3.68</td>
<td>1.76</td>
<td>9.08</td>
</tr>
<tr>
<td>mozilla</td>
<td>03/98</td>
<td>40545</td>
<td>8370</td>
<td>595</td>
<td>7.77</td>
<td>1.90</td>
<td>21.11</td>
</tr>
<tr>
<td>NetBSD</td>
<td>03/93</td>
<td>19514</td>
<td>7081</td>
<td>267</td>
<td>6.48</td>
<td>1.66</td>
<td>18.00</td>
</tr>
<tr>
<td>openssh</td>
<td>09/99</td>
<td>289</td>
<td>138</td>
<td>50</td>
<td>5.32</td>
<td>2.21</td>
<td>35.56</td>
</tr>
<tr>
<td>postgresql</td>
<td>07/96</td>
<td>4102</td>
<td>904</td>
<td>25</td>
<td>4.53</td>
<td>1.92</td>
<td>42.00</td>
</tr>
<tr>
<td>python</td>
<td>08/90</td>
<td>4643</td>
<td>419</td>
<td>88</td>
<td>5.94</td>
<td>1.94</td>
<td>31.59</td>
</tr>
</tbody>
</table>
Table 2: Estimations (OLS) for APACHE and ALL-WEIGHTED benchmark. 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 / $ < 0.05 / # < 0.10

<table>
<thead>
<tr>
<th></th>
<th>APACHE</th>
<th>ALL WEIGHTED</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Max maint's</td>
<td>Max revisions</td>
</tr>
<tr>
<td>Intercept</td>
<td>2.53270***</td>
<td>0.12740</td>
</tr>
<tr>
<td></td>
<td>2.41333***</td>
<td>0.22860</td>
</tr>
<tr>
<td>LOCs</td>
<td>-1.25E-4</td>
<td>7.47E-5</td>
</tr>
<tr>
<td></td>
<td>5.74E-4</td>
<td>2.96E-3</td>
</tr>
<tr>
<td>McCabe</td>
<td>8.64E-3*</td>
<td>3.65E-2</td>
</tr>
<tr>
<td></td>
<td>4.12E-3</td>
<td>2.12E-2</td>
</tr>
<tr>
<td>Halstead</td>
<td>1.25E-5</td>
<td>5.11E-5</td>
</tr>
<tr>
<td></td>
<td>1.70E-5</td>
<td>8.76E-5</td>
</tr>
<tr>
<td>Funs</td>
<td>4.80E-2***</td>
<td>0.1874*</td>
</tr>
<tr>
<td></td>
<td>1.15E-2</td>
<td>1.76E-2</td>
</tr>
<tr>
<td>Max maint's</td>
<td>2.41333***</td>
<td>0.22860</td>
</tr>
<tr>
<td>Youth</td>
<td>-2.91E-2***</td>
<td>-4.98E-2*</td>
</tr>
<tr>
<td></td>
<td>3.43E-3</td>
<td>1.76E-2</td>
</tr>
</tbody>
</table>

Adj. $R^2$            | 0.4134          | 0.3364          | 0.4808        | 0.3354          | 0.3348          | 0.5678          |
Table 3: Econometric estimations (OLS) for all 10 projects.
Dependent variables: maximum number of maintainers per month.
Between parenthesis, the sign of the coefficient.
Confidence levels: *** < 0.0001 / ** < 0.001 / * < 0.01 / $ < 0.05 / $ < 0.10

<table>
<thead>
<tr>
<th></th>
<th>Adj. R²</th>
<th>LOCs</th>
<th>McCabe</th>
<th>Halstead</th>
<th>Functions</th>
<th>Youth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>0.41</td>
<td>(+)$</td>
<td>(+)***</td>
<td>(-)***</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCVS</td>
<td>0.52</td>
<td>(+)***</td>
<td>(+)*</td>
<td>(-)***</td>
<td>(-)***</td>
<td></td>
</tr>
<tr>
<td>Gaim</td>
<td>0.46</td>
<td>(+)**</td>
<td>(+)</td>
<td>(-)***</td>
<td>(-)***</td>
<td></td>
</tr>
<tr>
<td>GCC</td>
<td>0.56</td>
<td>(+)***</td>
<td>(+)***</td>
<td>(-)***</td>
<td>(+)***</td>
<td>(-)***</td>
</tr>
<tr>
<td>Ghostscript</td>
<td>0.58</td>
<td>(-)***</td>
<td>(+)***</td>
<td></td>
<td>(-)***</td>
<td></td>
</tr>
<tr>
<td>Mozilla</td>
<td>0.28</td>
<td>(+)***</td>
<td>(+)</td>
<td>(+)***</td>
<td>(-)***</td>
<td></td>
</tr>
<tr>
<td>NetBSD</td>
<td>0.30</td>
<td>(+)***</td>
<td>(+)***</td>
<td>(-)*</td>
<td>(-)***</td>
<td></td>
</tr>
<tr>
<td>OpenSSH</td>
<td>0.67</td>
<td>(+)$</td>
<td>(+)$</td>
<td>(-)***</td>
<td>(-)***</td>
<td></td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>0.37</td>
<td>(+)***</td>
<td>(+)$</td>
<td>(-)***</td>
<td>(-)***</td>
<td></td>
</tr>
<tr>
<td>Python</td>
<td>0.56</td>
<td>(+)***</td>
<td>(-)***</td>
<td>(-)***</td>
<td>(-)***</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>0.38</td>
<td>(+)***</td>
<td>(+)***</td>
<td>(-)***</td>
<td>(+)***</td>
<td>(-)***</td>
</tr>
<tr>
<td>ALL WEIGHTED</td>
<td>0.34</td>
<td>(+)***</td>
<td>(+)***</td>
<td>(-)***</td>
<td>(+)***</td>
<td>(-)***</td>
</tr>
</tbody>
</table>