REVUE D'ÉCONOMIE INDUSTRIELLE

**Revue d'économie industrielle**

**136 | 4ème trimestre 2011**
**L'économie du logiciel libre**

# Organisation of work in Open Source Projects: expended effort and efficiency

**Stefan Koch**

OpenEdition Journals

**Electronic version**
URL: https://journals.openedition.org/rei/5165
DOI: 10.4000/rei.5165
ISSN: 1773-0198

**Publisher**
De Boeck Supérieur

**Printed version**
Date of publication: 15 December 2011
Number of pages: 17-38
ISSN: 0154-3229

**Electronic reference**
Stefan Koch, "Organisation of work in Open Source Projects: expended effort and efficiency", *Revue d'économie industrielle* [Online], 136 | 4ème trimestre 2011, Online since 15 December 2013, connection on 02 June 2022. URL: http://journals.openedition.org/rei/5165 ; DOI: https://doi.org/10.4000/rei.5165

Stefan KOCH
Department of Management, Bogazici University

# ORGANISATION OF WORK IN OPEN SOURCE PROJECTS : EXPENDED EFFORT AND EFFICIENCY

**Key words :** Open Source, Effort Estimation, Efficiency, Software Quality, Self-organization, Software Repository Mining.

**Mots-clés :** « Open Source », estimation de l'effort, éfficacité, qualité du logiciel, auto-organisation, « Software Repository Mining ».

## I. — INTRODUCTION

In the last years, open source software, *i.e.* software under a license that grants several rights like free redistribution to the user (1), has become more and more important, with this importance now stretching beyond the mere use of well-known projects in both private and commercial settings (Fitzgerald, 2006 ; von Krogh and Spaeth, 2007). Examples for this type of adoption naturally include the operating system Linux with the utilities of the GNU project and various desktops as well as office packages, the Apache web server, data bases such as MySQL and many others.

More importantly, open source software is of special interest due to its development process and organization of work. In many ways it can be seen as constituting a new production mode, in which people are no longer collocated, and self-organization is prevalent. The seminal work on this topic was written

(1) It should be noted that several terms are in use within this field, most notably open source software (Perens, 1999) and free software (Stallman, 2002). In this paper, the term open source is used to refer to free software as well, if a particular license is of importance, this is mentioned explicitly.

by Eric S. Raymond, « The Cathedral and the Bazaar », in which he contrasts the traditional type of software development of a few people planning a cathedral in splendid isolation with the new collaborative bazaar form of open source software development (Raymond, 1999). In this, a large number of developer-turned users come together without monetary compensation to cooperate under a model of rigorous peer-review and take advantage of parallel debugging that leads to innovation and rapid advancement in developing and evolving software products. In order to allow for this to happen and to minimize duplicated work, the source code of the software needs to be accessible, which necessitates suitable licenses, and new versions need to be released in short cycles.

This theoretical work is going to be our starting point in examining the organization of work in open source projects, and the implications of this organization on effort and efficiency as well as quality. The research on similarities and dissimilarities between open source development in general and other commercial software development models is still proceeding (Mockus *et al.,* 2002 ; Koch, 2004 ; Scacchi *et al.,* 2006), and remains a hotly debated issue (Bollinger *et al.,* 1999 ; McConnell, 1999 ; Vixie, 1999).

It is necessary to base such a discussion on empirical assessments of work practices and organizations in real-world projects. Empirical research on open source processes often employs the analysis of data available through mining the communication and coordination tools and their repositories. For this paper, we will mostly focus on this approach and results from several studies using it. Other approaches taken include (n)ethnographic studies of development communities (Coleman and Hill, 2004 ; Elliott and Scacchi, 2004), sometimes coupled with repository mining (Basset, 2004).

The structure of this paper is as follows : we will start with a short introduction to mining software repositories, and then provide a discussion and empirical data concerning the organization of work in open source projects based on this methodology. Then we are going to focus on the implications that these organizational aspects have on effort, as well as efficiency and quality. The paper will conclude with a synthesis and some comments on possible future work.

## II. — SOFTWARE REPOSITORY MINING

Software development repositories contain a plethora of informations on software and underlying, associated development processes (Cook *et al.,* 1998 ; Atkins *et al.,* 1999). Studying software systems and development processes using these sources of data offers several advantages (Cook *et al.,* 1998) : it is very cost-effective, as no additional instrumentation is necessary, and does not influence the process under consideration. In addition, longitudinal data are available, allowing for analyses that consider the whole project

history. Depending on the tools used in a project, possible repositories available for analysis include source code versioning systems (Atkins *et al.,* 1999 ; Kemerer and Slaughter, 1999), bug reporting systems, and mailing lists. In open source projects, repositories in several forms are also in use, and in fact form the most important communication and coordination channels as participants are not collocated. Therefore, there is only a small amount of information that cannot be captured because it is transmitted inter-personally. Repositories in use must also be available openly, in order to enable persons to access them and participate in the project.

Prior studies have included both in-depth analyses of small numbers of successful projects (Gallivan, 2001) like Apache and Mozilla (Mockus *et al.,* 2002), GNOME (Koch and Schneider, 2002) or FreeBSD (Dinh-Tong and Bieman, 2005), and also large data samples, such of those derived from Sourceforge.net (Koch, 2004 ; Long and Siau, 2007). Primarily, information provided by version control systems has been used, but also aggregated data provided by software repositories (Crowston and Scozzi, 2002 ; Hunt and Johnson, 2002 ; Krishnamurthy, 2002), meta-information included in Linux Software Map entries (Dempsey *et al.,* 2002), or data retrieved directly from the source code itself (Ghosh and Prakash, 2000).

Although this data is available, the task is made more complicated by the large size and scope of the project repositories or code forges, and the heterogeneity of the projects being studied (Howison and Crowston, 2004 ; Robles *et al.,* 2009). Therefore, in the last years RoRs (« repository of repositories ») have been developed, which collect, aggregate, and clean the targeted repository data (Sowe *et al.,* 2007). Two examples are FLOSSMetrics and FLOSSmole. These RoRs usually hold data collected from project repositories, and some of them also store some analysis and metrics calculated on the retrieved data. The results (raw data, summary data, and/or analyses) will be stored in a database and accessible to the rest of the research community. The researcher therefore does not need to collect data independently.

## III. — ORGANIZATION OF WORK IN OPEN SOURCE PROJECTS

The topic of this paper is to analyze open source software development as a possible special case of organization of work. Often, this style of development is hypothesized to be a new production mode, lacking collocation of members as well as any centralized management. While there is one seminal description of the bazaar style of development by Raymond (1999) enforcing this view, it should be noted that open source projects do differ significantly in the processes they employ (Scacchi *et al.,* 2006), and that reality has been found to differ from this very theoretical description. For example, there are strict release processes in place in several open source projects (Jorgensen, 2001 ; Holck and Jorgensen, 2004), and a considerable level of commercial involvement in several areas (Henkel. 2006 ; Roberts *et al.,* 2006 ; O'Mahony and Bechky,

2008). Governance structures in general do exist, and O'Mahony and Ferraro (2007) specifically focused on the emergence of governance, finding that members develop a shared basis of formal authority but limit it with democratic mechanisms that enabled experimentation with shifting conceptions of authority over time. Barcellini *et al.* (2008b) report on a study of the design dynamics in open source projects using an analysis of electronic discussions. They find community consensus as well as implicit rules to govern some of these exchanges, as well as specific participants (« top hierarchy ») active in framing. In a second, related study (Barcellini *et al.,* 2008a), the authors find several key participants acting as boundary spanners between user and developer communities (and mailing lists). They therefore argue that OSS design may be considered as a form of « role emerging design », *i.e.* design organized and pushed through emerging roles and through a balance between these roles, with the communities providing a suitable socio-technical environment to enable such role emergence.

Several ways have been discussed to describe different open source development processes, *e.g.* Crowston *et al.* (2006) operationalize a process characteristic based on the speed of bug fixing, Michlmayr (2005) used a construct of process maturity, while also concentration indices have been used to characterize development forms (Koch and Neumann, 2008). We find that there is considerable variance in the practices actually employed, as well as the technical infrastructure. It has been hypothesized that the advent of the Internet and especially the coordination and communications tools are at least a precondition for this development (Rusovan *et al.,* 2005 ; Robbins, 2005). For example, Raymond (1999) writes, « Provided the development coordinator has a medium at least as good as the Internet, and knows how to lead without coercion, many heads are inevitably better than one ». Several studies have especially focused on mailing list discussions surrounding open source projects (Kuk, 2006 ; Barcellini *et al.,* 2008a, 2008b). On a conceptual level, Hemetsberger and Reinhardt (2009) in their study of KDE draw on the concept of co-configuration, which is is a participatory model that is not confined to collaboration between professionals, and integrates users as active subjects, who are active in the shaping and reshaping of products and eventually become experts themselves. The use the term of coat-tailing work systems that tie everyday actions to the overall activity of the group, and underscore the importance of technological, cultural and mental artefacts to construct such a system so that online collaboration can transgress the limitations of the dispersed group work.

Numerous quantitative studies of development projects and communities (Dempsey *et al.,* 2002 ; Dinh-Trong and Bieman, 2005 ; Ghosh and Prakash, 2000 ; Koch and Schneider, 2002 ; Koch, 2004 ; Krishnamurthy, 2002 ; Mockus *et al.,* 2002) have proposed process metrics like the commit, which refers to a single change of a file by a single programmer, or the number of distinct programmers involved in writing and maintaining a file or project to study open source work practices. One of the most consistent results coming out of this

research is a heavily skewed distribution of effort between participants (Koch, 2004 ; Mockus *et al.,* 2002 ; Ghosh and Prakash, 2000 ; Dinh-Tong and Bieman, 2005). Several studies have adopted the normalized Gini coefficient (Robles *et al.,* 2004), a measure of concentration, for this. The Gini coefficient is a number between 0 and 1, where 0 is an indicator for perfect equality and 1 for total inequality or concentration, and can be based both on commits or lines-of-code contributed, with studies showing no major difference. For example, Mockus *et al.* (2002) have shown that the top 15 of nearly 400 programmers in the Apache project added 88 per cent of the total lines-of-code. In the GNOME project, the top 15 out of 301 programmers were only responsible for 48 percent, while the top 52 persons were necessary to reach 80 per cent (Koch and Schneider 2002), with clustering hinting at the existence of a still smaller group of 11 programmers within this larger group. A similar distribution for the lines-of-code contributed to the project was found in a community of Linux kernel developers by Hertel *et al.* (2003). Also the results of the Orbiten Free Software survey (Ghosh and Prakash 2000) are similar, the first decile of programmers was responsible for 72 per cent, the second for 9 per cent of the total code.

A second major result regarding organization of work is a low number of people working together on file level. For example, Koch and Neumann (2008) have found that only 12.2 % of the files have more than three distinct authors. Most of the files have one (24.0 %) or two (56.1 %) programmers and only 3 % have more than five distinct authors, in accordance with other studies on file or project level (Koch, 2004 ; Krishnamurthy, 2002 ; Mockus *et al.,* 2002 ; Ghosh and Prakash, 2000).

Similar distribution can also be found on project level in large scale studies : for example, Koch (2004) in his study of several thousand projects found a vast majority of projects having only a very small number of programmers (67.5 per cent have only 1 programmer). Only 1.3 per cent had more than 10 programmers. Analyzing the 100 most active mature projects on Sourceforge.net, Krishnamurthy (2002) also showed that most of the projects had only a small number of participants (median of 4). Only 19 per cent had more than 10, 22 per cent had only 1 developer. While this percentage is much smaller than found by Koch (2004), this is not surprising as Krishnamurthy only used the 100 most active projects, not the full population.

Another aspect that cannot be underestimated with regard to the implications for organization of work is the increased commercial interest in open source software. This has also lead to changes in many projects, which now include contributors who get paid for their contributions and others, who receive no direct payment. This can have repercussions on motivation and participation (Roberts *et al.,* 2006), and is also reflected in several surveys : for example, Lakhani and Wolf (2005) found that 13 % of respondents received direct payments, and 38 % spent work hours on open source development with their supervisor being aware of the fact. Ghosh (2005) reports a group of 31.4 %

motivated by monetary or career (mostly for signaling competence) concerns in a sample of 2,280 responses. Hars and Ou (2001) found a share of 16 % being directly paid, Hertel *et al.* (2003) report 20 % of contributors receiving a salary for this work on a regular basis with an additional 23 % at least sometimes in a survey of Linux kernel developers. Demetriou *et al.* (2007) have shown some implications of this fact in a case study of the OpenACS project : historically, developers with a commercial interest dominated the project history and code base, but this fact might be slowly changing, with his large amount of commercial interest having led to a governance structure which puts great value on control and stability by requiring Technical Improvement Proposals for major changes. On the other hand, this rigidity seems to have affected the way of work, in that sideway developments might be established creating coexisting sub-frameworks. From an architectural viewpoint, this would be disadvantageous, and it might also have the effect of preventing true open source style development, as the code in these parts would tend to be more specific and only usable in a certain context. In the empirical data, there seem to be indications for this happening especially in conjunction with commercial developers : the authors found that packages being to a high degree dominated by commercial background tend to include less developers overall and less volunteers, and also tend to be changed less often and by the same group of people.

On a larger scale, the interactions of open source projects and commercial entities in open innovation activities have been an active field of research for several years. Henkel (2006) has analysed the behaviour of companies with regard to free revealing in the context of embedded Linux, O'Mahony and Bechky (2008) focus on the role of boundary spanners in achieving collaboration between open source projects and firms. We will not consider this organization-level cooperation and collaboration issues in this paper, but limit the discussion to intra-project phenomena.

After this presentation of some empirical results concerning the organization of work within open source projects, we will now turn to the relation with effort, as well as with efficiency and quality.

## IV. — ORGANIZATION OF WORK AND EFFORT

### 1. Effort estimation for open source projects

We focus now on the effort expended for open source projects, with the main goal of uncovering relationships between this effort and the open source development model overall, *i.e.* comparing it to other – commercial – approaches of organizing work, as well as different form of organization within projects. Any discussion on open source projects, their organization, success, efficiency or quality in general is incomplete without addressing this topic of effort, and relating other constructs to it. Unfortunately, this effort is basically unk-

nown, even to the leaders of these projects, and therefore needs to be estimated. In addition such an estimation offers important insights to stakeholders, *e.g.* in the context of decisions about how an ongoing project might be managed, whether to join or to remain in a project, as well as adoption decisions by current or prospective users, including companies deciding whether to pursue a related business model. It will allow a most basic measure for the value created by such communities.

Software engineering research for many years has focused on the topic of effort estimation, and has produced numerous models and methods. The best known of these is probably COCOMO (Boehm, 1981), which offers an algorithmic formula for estimating effort based on a quantification of the lines of code ; this was modified and updated with the publication of COCOMO II (Boehm *et al.,* 2000). Both rely on a Cobb-Douglas production formula based on a regression performed on a set of (commercial) projects, merged with expert judgement. In the actual use, the main cost driver is software size, influenced by an assessment of other influence factors like development flexibility or participant experience. Other options for effort estimation include the software equation (Putnam, 1978), approaches based on the function point metric (Albrecht and Gaffney, 1983), diverse machine-learning approaches and proprietary models such as ESTIMACS and SLIM. Many of these approaches are based on the general development project formulation introduced by Norden (1960), which develops a manpower function based on the number of people participating in a project at a given time. Differences in work organization in open source projects raise the question of whether participation in OS projects can be modeled and predicted using approaches created in the context of traditional software development, or whether new models need to be developed. Naturally this does not apply to machine-learning models, but those would require a sufficient database of similar projects with known efforts, which is not available in this context.

In general, we can employ two different approaches to estimating effort, one based on output, *i.e.* software and some measure of its size, the other based on evidence of participation. Especially the comparison of both approaches can yield important insights related to comparing open source to commercial software development, as the estimation based on size basically assumes an organization equivalent to commercial settings. In this paper, we will mostly focus on the results of this process, while for a more in-depth coverage, the reader is referred to Koch (2008), where the analysis is based on a set of more than 8000 projects. Other related work is limited (Koch and Schneider, 2002 ; Koch, 2004 ; Gonzalez-Barahona *et al.,* 2004 ; Wheeler, 2005 ; Amor *et al.,* 2006), and for the most part applies only basic models without further discussion, or indirect effort measures (Yu, 2006).

For participation-based estimation, the basis is formed by the work of Norden (1960), which models any development project as a series of problem-solving efforts by the manpower involved. The manpower involved at each

moment in an open source project can be inferred from an analysis of the source code management system logs. The number of people usefully employed at any given time is assumed to be approximately proportional to the number of problems ready for solution at that time. The manpower function then represents a Rayleigh-type curve. While Norden postulates a finite and fixed number of problems, additional requests would lead to the generation of new problems to be worked on. While this effect might be small to negligible until the time of operation, it might be a driving factor later on. Therefore, while there are similarities in the early stages of a project, in the later stages, distinct differences in processes and organization of work show up, linked to differences in goal setting and eliciting. We will therefore further explore this possible effect of work organization in the next section.

The first model in the output-based estimation category is the original COCOMO (Boehm, 1981), and while severe problems related to use of this model due to violated assumptions exist, it is still employed for comparison with other models and with existing studies (Gonzalez-Barahona *et al.,* 2004 ; Wheeler, 2005). The necessary data on software size can easily be gathered from the source code management system of any open source project, or by downloading the source code itself and submitting it to a counting program. The updated version COCOMO 2 (Boehm *et al.,* 2000) eliminates some of these concerns, so forms another option. An approach that is similar to COCOMO in that it is also based on an output metric, is the function point method (Albrecht and Gaffney, 1983). This method in general offers several advantages, most importantly the possibility to quantify the function points relatively early, based on analysis and design. In estimating the effort it is difficult, especially for an outsider, to correctly quantify the function points, even after delivery. Another way of arriving at a number is to use the converse method of converting the function point count to lines-of-code (Albrecht and Gaffney, 1983 ; Boehm *et al.,* 2000). The literature proposes a mean number of lines-of-code required to implement a single function point in a given programming language. Once the amount of function points for a given system is known, literature provides several equations, basically production functions, to relate this amount to effort, naturally all based on data collected in a commercial software development environment. Examples include Albrecht and Gaffney (1983), Kemerer (1987) or Matson *et al.* (1994), who propose both a linear and logarithmic model.

## 2. Organization and effort

We first explore any differences that the particular style of organization of work in open source projects could mean for manpower modeling based on the work of Norden (1960). As discussed, while Norden postulates a finite and fixed number of problems, additional requests from the user and developer community could lead to the generation of new problems to be worked on. Based on extensive modeling and comparisons using several alternative manpower functions for a set of more than 8000 projects (Koch, 2008), modified

Norden-Rayleigh-functions incorporating this effect significantly outperform the classical variant over complete project lifespans. We can see this as proof that in the open source form of organization, additional requirements and functionalities are introduced to a higher degree than in commercial settings. Also several possible forms for adding this effect to the Norden-Rayleigh model have been explored (Koch, 2008) : the features added seem to depend on the starting problem to a higher extent than on the cumulative effort expended up to that time. This highlights the importance of the first set of requirements, often the vision of the project founder, in determining future enhancements. Also, a different proportionality factor, *i.e.* learning rate, has to be assumed as compared to the main respectively initial project, with a quadratic function better suited to modeling the addition of new problems. These results underline how much open source software development is actually driven by the participants and users, who truly shape the direction of such a project according to their needs or ideas.

We next turn to an analysis of the differences between participation-based estimation and output-based estimation. As the approaches in the latter group are all based on data from commercial settings, these differences can give an idea about the relation between participation-based effort, and the effort that would be necessary to develop the same system in a different environment and using a different organization. The empirical analysis (Koch, 2008) shows distinctive differences : estimates derived from Norden-Rayleigh modeling were tested against each output-based method, and were significantly lower. An analytical comparison is also possible between COCOMO in both versions (Boehm, 1981 ; Boehm *et al.,* 2000), and the Norden-Rayleigh model because COCOMO is based on this. Londeix (1987) detailed how the Rayleigh-curve corresponding to a given COCOMO estimation can be determined. In this case the other direction is employed to find a parameter set in COCOMO corresponding to the Rayleigh-curve, derived from programmer participation. As COCOMO offers both development mode and a number of cost drivers as parameters, there is no single solution. Nevertheless, actually not a single solution is possible given the parameter space, so open source development cannot be modeled using the original COCOMO, which leads to the conclusion that development has been more efficient than theoretically possible. When the possible parameters of COCOMO II are explored, the result is once again that this project is very efficient as both cost drivers and scale factors replacing the modes of development in COCOMO II have to be rated rather favorably, but this time the resulting combinations are within the range.

These differences showing up between the effort estimates based on participation and output might be due to several reasons. First, open source development organization might constitute a more efficient way of producing software (Bollinger *et al.,* 1999 ; McConnell, 1999), mostly due to self-selection outperforming management intervention. Participants might be able to determine more accurately whether and where they are able to work productively on the project overall, or on particular tasks. This could also be relabeled as an indi-

cation of the success of coat-tailing work systems to tie everyday actions to the overall activity of the group (Hemetsberger and Reinhardt, 2009). In addition, overhead costs are very much reduced. The second explanation might be that the difference is caused by non-programmer participation, *i.e.* people participating by discussing on mailing lists, reporting bugs, maintaining web sites and the like. These are not included in the participation-based manpower modeling due to the fact that data is based on activities within the source code management systems. If the Norden-Rayleigh and COCOMO estimates are compared, COCOMO results for effort are eight times higher. If it were assumed that this difference could only come from the invisible effort expended by these participants, this effort must be enormous. It would account for about 88 percent of the effort, translating to about seven persons assisting each programmer. As has been shown, in open source projects, the number of participants other than programmers is about one order of magnitude larger than the number of programmers (Dinh-Trong and Bieman, 2005 ; Mockus *et al.,* 2002 ; Koch and Schneider, 2002), but their expended effort is implicitly assumed to be much smaller. We are therefore going to relate this to the the « chief programmer team organization », proposed more than 30 years ago (Mills, 1971 ; Baker, 1972). This has also been termed the « surgical team » by Brooks (1995), and is a form of organization where which system development is divided into tasks each handled by a chief programmer who has responsibility for the most part of the actual design and coding, supported by a larger number of other specialists such as documentation writers or testers. We are going to expand on this discussion in the conclusion, using additional results derived from research in to efficiency.

## V. — ORGANIZATION OF WORK AND EFFICIENCY

### 1. Efficiency of open source projects

In this part, we will explore the effects of organization of work in open source projects on effficiency and quality within those projects. Conceptualising efficiency of open source projects is actually more difficult for open source projects. In software development, efficiency or productivity is most often denoted by the relation of an effort measure to an output measure, using either lines-of-code or, preferably due to independence from programming language, function points (Albrecht and Gaffney, 1983). This approach can be problematic even in an environment of commercial software development due to missing components especially of the output, for example also Kitchenham and Mendes (2004) agree that productivity measures need to be based on multiple size measures.

In open source development, the effort invested as discussed is normally unknown and consequently needs to be estimated, and the participants are also more diverse than in commercial projects as they include core team member, committers, bug reporters and several other groups with varying intensity of

participation. Besides, also the outputs can be more diverse. Many researchers have worked on conceptualising the success of such projects (Stewart, 2004 ; Stewart *et al.,* 2006 ; Stewart and Gosain, 2006 ; Crowston *et al.,* 2003 ; Crowston *et al.,* 2004 ; Crowston *et al.,* 2006), which can provide a starting point as well for a set of output measures. Literature has thus suggested a huge number of indicators, using, for example, search engine results as proxies (Weiss, 2005), or measures like number of downloads achieved, or even subjective answers to surveys (Stewart and Gosain, 2006), but aggregating those to have an overall picture has been a major problem. In the general case, the inputs of an open source project can encompass a set of metrics, especially concerned with the participants. In the most simple cases the number of programmers and other participants can be used. The output of a project can be measured using several software metrics, most easily the lines-of-code, files, or others like downloads achieved. This range of metrics both for inputs and outputs, and their different scales necessitates the application of an appropriate method. Many of the results presented in the next two sections are therefore based on applying Data Envelopment Analysis (DEA) to this problem. DEA (Farell, 1957 ; Charnes *et al.,* 1978 ; Banker *et al.,* 1984) is a non-parametric optimization method for efficiency comparisons without any need to define any relations between different factors or a production function. In addition, DEA can account for economies or dis-economies of scale, and is able to deal with multi-input, multi-output systems in which the factors have different scales.

The main result of applying DEA for a set of projects is an efficiency score for each project. This score can serve different purposes : first, single projects can be compared accordingly, but also groups of projects, for example those following similar process models, located in different application domains or simply of different scale can be compared to determine whether any of these characteristics lead to higher efficiency.

## 2. Organization and efficiency

In this section, we will give an overview of the interrelationship between different attributes of open source projects characterizing their organization of work, as well as infrastructure, and their efficiency.

The first element to be explored naturally is the generally large number of participants. Following the reasoning of Brooks, an increased number of people working together will decrease productivity as measured by lines-of-code produced per participant per time intervall due to exponentially increasing communication costs (Brooks, 1995). Interestingly, this effect has not turned up in prior studies (Koch, 2004 ; Koch, 2007). This leads to the interesting conclusion that Brooks's Law seemingly does not apply to open source software development. There are several possible explanations for this, which include the very strict modularization, which increases possible division of labor while reducing the need for communication. Also the low number of pro-

grammers working together on single files can be taken as a hint for this. We will also explore the notion of superior tool and infrastructure use as a possible factor later.

As empirical results have shown that the effort within open source projects is distributed very inequally, which seems to be a major characteristic of this type of organization, any effects this could have on efficiency should also be explored. Using a data set of projects from SourceForge and DEA, Koch (2008a) showed that there is indeed no connection : there was no significant difference in efficiency between projects with different levels of inequality, so this form of organization does not seem to incur a penalty. In some works, also license, especially GNU GPL, is hypothesized as having an impact on success or efficiency. Subramanian *et al.* (2009) found such an effect, as did Stewart *et al.* (2006), while Koch (2008a) did not.

Closely linked to efficiency in a project is it's rate of progress, which, although especially growth in participant numbers provides another major contributing factor as well, is affected by it. The evolution and development speed of commercial systems has been an issue that has long been a center of research, and therefore a coherent theoretical framework of software evolution has been developed and empirically tested, mainly based on the works of Lehman and others (Belady and Lehman, 1976 ; Lehman and Ramil, 2001). For open source software systems, several works have explicitly dealt with this topic. The first and one of the most important contributions is a case study by Godfrey and Tu (2000), who have analyzed the most prominent example available, the Linux operating system kernel. They found that the growth behavior is best fitted by a super-linear rate, which contradicts the prior theory of software evolution, which postulates a decline in growth. On the other hand, Paulson *et al.* (2004) have used a linear approximation, and have not found any differences in growth behavior between open and closed-source software projects. Robles *et al.* (2003) reproduced the study of Godfrey and Tu (2000) with newer data, and found similar results, in addition showing that the growth of Linux has even accelerated during the five years between both works. They have also analyzed major subsystems, finding also super-linear growth patterns on this level. In addition, they surveyed 18 more large open source projects (like Apache, GNOME or KDE), finding growth patterns linear or close to linear for 16 of them, with the other two exhibiting some special characteristics. In another case study, Robles *et al.* (2006) found that the KDE project shows super-linear growth. Capiluppi *et al.* (2003) presented the first large horizontal study of open source system evolution using 406 projects from a repository, with size constantly growing in active projects. They also note that in large and medium projects, the number of modules grows, but their size tends to evolve to a stable value. Finally, Scacchi (2004) gives a discussion of open source software evolution, with an overview and review of studies both on proprietary and open source projects. He concludes that the laws of software evolution do not account for the potential for super-linear growth in software size that can be sustained by satisfied developer-user communities. Koch

(2007) uses a large sample of projects, and comes to the conclusion that while in the mean the growth rate is linear or decreasing over time according to the laws of software evolution, a significant percentage of projects is able to sustain super-linear growth. There is a positive relationship between the size of a project, the number of participants and the inequality in the distribution of work within the development team with the presence of super-linear growth patterns. On the other hand, there is evidence for a group of projects of moderate size which shows decreasing growth rates, while small projects in general exhibit linear growth. A possible explanation for this fact is that projects with a super-linear growth rate are able to implement a certain organizational model, the chief programmer team, as mentioned above. This will be revisited in the conclusions. This form, present in open source software development, through measures like strict modularization and self-selection for tasks (Crowston *et al.,* 2007) seems to be able to at least delay the negative effects arising during evolution. In addition, especially the fourth law of software evolution, « conservation of organizational stability » (Lehman & Ramil, 2001), implying constant incremental effort, might be violated especially in very large and prominent projects which attract an ever increasing number of participants.

Finally, the infrastructure employed for communication and coordination naturally shapes the work done in a project. It has been hypothesized that the advent of the Internet and especially the coordination and communications tools are at least a precondition for open source development (Raymond, 1999 ; Rusovan *et al.,* 2005 ; Robbins, 2005). For example, Michlmayr (2005) has used a sample of projects to uncover whether the process maturity, based on version control, mailing lists and testing strategies, has had any influence on the success of open source projects, and could confirm this. Koch (2009) has analyzed the impact of adoption of different tools offered by SourceForge as well as tool diversity on project efficiency using DEA, and found surprising results : in a data set of successful projects, actually negative influences of tool adoption were found, while the results were more positive in a random data set. Two explanations were proposed for this, with one being that projects, especially larger ones, might be using other tools. The second explanation is that tools for communicating with users and potential co-developers can become more of a hindrance in successful projects, as they could increase the load to a degree that it detracts attention and time from the developers, which might be better spent on actual development work. In general, the successful projects also show a more progressed status, so actually these results seem to correspond to the results of Stewart and Gosain (2006), who stress the importance of development stage as moderator in project performance. In addition, these projects in general have a higher number of developers, which, counter-intuitively, seems to be linked to negative effects of communication and coordination tool adoption. One explanation might be that projects with problems in communication and coordination due to team size adopt tools to a higher degree, which can not completely solve the problem after it has passed some threshold. Therefore projects adopting these tools have a lower efficiency, but that might

be even lower without tool adoption. The same reasoning could apply for communication channels with users : tool adoption alone might be unable to prevent total communication overload. A third additional explanation put forward here links tool adoption to an increase in user involvment, which in turn might influence project design and fit with user expectations. This increased fit could also lead to an actual reduction in planned functionality, if it does not add value to the user base, as well as optimization in code use. Therefore, due to the increased user integration offered through higher tool adoption, the size of the software, which is one output aspect considered in deriving the efficiency measure, could actually be reduced, leading to a perceived lower efficiency, although in reality a higher fit with user requirements and code optimization has been achieved.

## 3. Organization and quality

In addition to effects of organization on efficiency, quality is a major concern in software development, and a hugely debated topic in open source (Dinh-Trong and Bieman, 2005 ; Stamelos *et al.,* 2002 ; Zhao and Elbaum, 2000). We will therefore highlight a few results which link elements of organization to the quality achieved, although related studies are quite rare. For capturing this, attributes of the development process as used before need to be related to characteristics of quality for which diverse metrics from software engineering like McCabe's cyclomatic complexity (McCabe, 1976) or Chidamber and Kemerer's object-oriented metrics suite (Chidamber and Kemerer, 1994) can be employed. All of those try to capture different aspects related to coding and design quality, *e.g.* complexity of code fragments, which in turn have been shown to have a major impact on aspects like maintainability, as well as the occurrence of bugs in the software or parts of the software. Koch and Neumann (2008) have attempted such an analysis using Java frameworks, and found that a high number of programmers and commits, as well as a high concentration is associated with problems in quality on class level, mostly to violations of size and design guidelines, thus being related to higher bug counts as well as problems in maintenance. This underlines the results of Koru and Tian (2005), who have found that modules with many changes rate quite high on structural measures like size or regarding inheritance. If the architecture is not modular enough, a high concentration might show up as a result of this, as it can preclude more diverse participation. The other explanation is that classes that are programmed and/or maintained by a small core team are more complex due to the fact that these programmers « know » their own code and don't see the need for splitting large and complex methods. One possibility in this case is a refactoring (Fowler, 1999) for a more modular architecture with smaller classes and more pronounced use of inheritance. This would increase the possible participation, thus maybe in turn leading to lower concentration, and maintainability together with other quality aspects. Underlining these results, MacCormack *et al.* (2006) have in a similar study used design structure matrices to study the difference between open source and proprietary developed software, without further discrimination in development practices. They

find significant differences between Linux, which is more modular, and the first version of Mozilla. The evolution of Mozilla then shows purposeful redesign aiming for a more modular architecture, which resulted in modularity even higher than Linux. They conclude that a product's design mirrors the organization developing it, in that a product developed by a distributed team such as Linux was more modular compared to Mozilla developed by a collocated team. Alternatively, the design also reflects purposeful choices made by the developers based on contextual challenges, in that Mozilla was successfully redesigned for higher modularity at a later stage.

On project level, there is a distinct difference (Koch and Neumann, 2008) : those projects with high overall quality ranking have more authors and commits, but a smaller concentration than those ranking poorly. Thus, on class level a negative impact of more programmers was found, while on project level a positive effect. This underlines a central statement of open source software development on a general level, that as many people as possible should be attracted to a project. On the other hand, these resources should, from the viewpoint of product quality, be organized in small teams. Ideally, on both levels, the effort is not concentrated on too few of the relevant participants. Again, this seems to point to the organizational form of « chief programmer team organization » (Mills, 1971 ; Baker, 1972 ; Brooks, 1995).

One final topic that was covered in a previous study already mentioned (Demetriou *et al.,* 2007) can be put into this context as well : the involvement of developers with a commercial interest seems to have affected the way of work in the case study presented. Sideway developments might be established creating coexisting sub-frameworks, which would be disadvantageous from an architectural viewpoint. Packages dominated by commercial background tend to include less developers overall and less volunteers, and also tend to be changed less often and by the same group of people. Again, this is quite contrary to the open source development model, and could create serious problems related to quality as well as long-term maintainability.

## VI. — CONCLUSIONS AND FUTURE WORK : CHIEF PROGRAMMER TEAM ORGANISATION

In this paper, we have surveyed the available literature related to organization of work within open source projects, and implications for effort and quality. Most of the empirical works have been based on mining the associated software repositories, and the results show that this is a promising way of achieving insights into projects and their characteristics.

One of the main principles and results found is the high concentration of programming work on a small number of individuals, which seems to hold true for most projects, similar to a very skewed distribution between projects. In addition, the number of people working on files cooperatively is quite small,

with commercial involvement even increasing this trend. Under some circumstances, this high concentration can be linked to problems in quality and maintainability. The number of programmers attracted to a project forms a main focus point, and generally has positive implications : having more programmers, quite interestingly and contrary to software engineering theory, does not reduce productivity, and does not negatively affect quality, if the concentration is kept in check. When considering the effort, estimations based on programmer involvement are significantly below estimations based on project output. Besides high efficiency due to self-organization and absence of management overhead, this points to an enormous effort expended by non-programming participants.

Many of these results point to one especially important characteristic that determines success of an open source projects, which is modularity. A modular architecture allows for high participation while avoiding the problems of high concentration on a lower level like file or class. The relation between core programmers and other participants also points on a certain organizational model on lower level, already proposed more than 30 years ago, the « chief programmer team organization » (Mills, 1971 ; Baker, 1972). This has also been termed the « surgical team » by Brooks (1995) after the model of a chirurgical team, and is a form of organization where system development is divided into tasks each handled by a chief programmer who has responsibility for the most part of the actual design and coding, supported by a larger number of other specialists such as documentation writers or testers. This model was never fully established in traditional software development environments, but seems to be successful in the context of open source projects. This can be attributed to the strict modularization, but also to the fact that within this different framework, volunteers are available that prefer to choose other roles than chief programmer, or maybe only have time resources or technical capabilities for filling those roles. In addition, given Internet technologies, it is much easier to form such teams and dissolve them again on an informal and ad-hoc basis. There is no hierarchy or fixed assignment between chief programmers and their support staff, but this is mitigated through self-selection of participants (Crowston *et al.,* 2007) and the modular architecture (MacCormack *et al.,* 2006) as well as the infrastructure employed, basically a coat-tailing work system (Hemetsberger and Reinhardt, 2009). Barcellini *et al.* (2008a) use the term of « role emerging design », *i.e.* design organized and pushed through emerging roles and through a balance between these roles, with the communities providing a suitable socio-technical environment to enable such role emergence. We therefore hypothesize that open source development is a form of production coalescing around a limited number of chief programmers that in changing constellations receive support from participants self-selected for these tasks based on interest and resources. The modular architecture allows, as well as follows from a higher number of such teams, and allows to scale this development mode until we arrive at the large and successful projects we know today.

There are many avenues for future research which are open in the context of open source software development, and a few have been touched upon here. The topic of effort and effort estimation is far from closed yet, and especially participants other than programmers are not adequately reflected so far. Also the relations between projects, for example inclusion of results or reuse, and linking this to efficiency, effort and organization, similar to a market versus hierarchy discussion would be highly interesting. There are numerous works that have used social network analysis, *e.g.* by Grewal *et al.* (2006) or Oh and Jeon (2007), both between as well as within projects, and this could be an interesting lens through which to inspect the chief programmer team aspect discussed here. Also Dalle and David (2005) have started to analyze the allocation of resources in projects. Finally, the conceptualization of success in the context of open source projects is still vague, and this uncertainty undermines some of the results from other studies or aspects.

Barcellini *et al.* (2008b) report on a study of the design dynamics in open source projects using an analysis of discussions. They find community consensus as well as implicit rules to govern some of these exchanges, as well as specific participants (« top hierarchy ») active in framing. In a second related study (Barcellini *et al.* 2008a), the authors find several key participants acting as boundary spanners between user and developer communities (and mailing lists). They therefore argue that OSS design may be considered as a form of « role emerging design », *i.e.* design organized and pushed through emerging roles and through a balance between these roles, with the communities providing a suitable socio-technical environment to enable such role emergence.

## REFERENCES


ALBRECHT A.J., GAFFNEY J.E. (1983), « Software Function, Source Lines of Code, and Development Effort Prediction : A Software Science Validation », *IEEE Transactions on Software Engineering,* 9(6), pp. 639-648.

AMOR J.-J., ROBLES G., GONZALEZ-BARAHONA J.-M. (2006), « Effort estimation by characterizing developer activity », in : *Proc. of the 2006 International Workshop on Economics Driven Software Engineering Research,* International Conference on Software Engineering, Shanghai, China, pp. 3-6.

ATKINS D., BALL T., GRAVES T., MOCKUS A. (1999), « Using Version Control Data to Evaluate the Impact of Software Tools », in : *Proc. 21st International Conference on Software Engineering,* Los Angeles, CA, pp. 324-333.

BAKER F.T. (1972), « Chief Programmer Team Management of Production Programming », *IBM Systems Journal,* 11(1), pp. 56-73.

BANKER R.D., CHARNES A. and COOPER W. (1984), « Some Models for Estimating Technical and Scale Inefficiencies in Data Envelopment Analysis », *Management Science,* 30, 1078-1092.

BARCELLINI F., DETIENNE F. and BURKHARDT J.-M. (2008a), « User and developer mediation in an Open Source Software community : Boundary spanning through cross participation in online discussions », *International Journal of Human-Computer Studies,* 66, pp. 558-570.

BARCELLINI F., DETIENNE F., BURKHARDT J.-M. and SACK W. (2008b), « A socio-cognitive analysis of online design discussions in an Open Source Software community », *Interacting with Computers,* 20, pp. 141-165.
BASSET T. (2004), « Coordination and social structures in an open source project : Videolan », in Koch, S., editor, *Open Source Software Development,* pp. 125-151. Idea Group Publishing, Hershey, PA.
BELADY L.-A. and LEHMAN M.-M. (1976), « A model of large program development », *IBM Systems Journal,* 15(3), pp. 225-252.
BOEHM B.W. (1981). *Software Engineering Economics.* Prentice Hall, Englewood Cliffs, NJ.
BOEHM B.W., ABTS C., BROWN A.W., CHULANI S., CLARK B.K., HOROWITZ E., MADACHY R., REIFER D.J., STEECE B. (2000), *Software Cost Estimation with COCOMO II.* Prentice Hall, Upper Saddle River, NJ.
BOLLINGER T., NELSON R., SELF K.M., TURNBULL S.J. (1999), « Open-source methods : Peering through the clutter », *IEEE Software,* 16(4), pp. 8-11.
BROOKS Jr. F.P. (1995), *The Mythical Man-Month : Essays on Software Engineering,* Anniversary ed., Addison-Wesley, Reading, MA.
CAPILUPPI A., LAGO P. & MORISIO M. (2003), « Evidences in the evolution of OS projects through Changelog Analyses », *Proceedings of the 3rd Workshop on Open Source Software Engineering,* pp. 19-24, 25th International Conference on Software Engineering, Portland, Oregon.
CHARNES A., COOPER W. and RHODES E. (1978), « Measuring the Efficiency of Decision Making Units », *European Journal of Operational Research*, 2, pp. 429-444.
CHIDAMBER S. and KEMERER C.-F. (1994), « A metrics suite for object oriented design », *IEEE Transactions on Software Engineering,* 20(6), pp. 476-493.
COLEMAN E. G. and HILL B. (2004), « The social production of ethics in debian and free software communities : Anthropological lessons for vocational ethics », in Koch, S., editor, *Open Source Software Development,* pp. 273-295. Idea Group Publishing, Hershey, PA.
COOK J.E., VOTTA L.G., WOLF A.L. (1998), « Cost-effective analysis of in-place software processes », *IEEE Transactions on Software Engineering,* 24(8), pp. 650-663.
CROWSTON K., SCOZZI B. (2002), « Open source software projects as virtual organizations : Competency rallying for software development », *IEE Proceedings – Software Engineering,* 149(1), pp. 3-17.
CROWSTON K., ANNABI H. and HOWISON J. (2003), « Defining Open Source Software Project Success », in *Proceedings of ICIS 2003,* Seattle, WA.
CROWSTON K., ANNABI H., HOWISON J. and MASANGO C. (2004), « Towards A Portfolio of FLOSS Project Success Measures », in *Collaboration, Conflict and Control : The 4th Workshop on Open Source Software Engineering (ICSE 2004),* Edinburgh, Scotland.
CROWSTON K., HOWISON J. and ANNABI H. (2006), « Information systems success in free and open source software development : theory and measures », *Software Process : Improvement and Practice,* 11(2), 123-148.
CROWSTON K., LI Q., WEI K., ESERYEL Y. and HOWISON J. (2007), « Self-organization of teams for free open source software development », *Information and Software Technology,* 49, pp. 564-575.
DALLE J.-M. and DAVID P.A. (2005), « The Allocation of Software Development ressources in Open Source Production Mode », in Feller, J., Fitzgerald, B., Hissam, S. A.i and Lakhani, K. R., editors, *Perspectives on Free and Open Source Software*, pp. 297-328. MIT Press, Cambridge, MA.
DEMETRIOU N., KOCH S. and NEUMANN G. (2007), « The Development of the OpenACS Community », in Lytras, M. and Naeve, A. (eds.) *Open Source for Knowledge and Learning Management : Strategies Beyond Tools,* Hershey, PA : Idea Group.
DEMPSEY B.J., WEISS D., JONES P., GREENBERG J., (2002), « Who is an open source software developer ?», *CACM*, 45(2), pp. 67-72.
DINH-TRONG T.-T., BIEMAN J.-M. (2005), « The FreeBSD Project : A Replication Case Study of Open Source Development », *IEEE Transactions on Software Engineering,* 31(6), pp. 481-494.
ELLIOTT M.-S. and SCACCHI W. (2004), « Free software development : cooperation and conflict in a virtual organizational culture », in Koch, S., editor, *Open Source Software Development,* pp. 152-172. Idea Group Publishing, Hershey, PA.

FARELL M.-J. (1957), « The Measurement of Productive Efficiency », *Journal of the Royal Statistical Society,* Series A 120(3), pp. 250-290.
FITZGERALD Brian (2006), « The Transformation of Open Source Software », *MIS Quarterly,* 30(3), pp. 587-598.
GALLIVAN Michael J. (2002), « Striking a balance between trust and control in a virtual organization : a content analysis of open source software case studies », *Information Systems Journal,* 11(4), pp. 277-304.
GHOSH R. A. (2005), « Understanding free software developers : findings from the floss study », in Feller, J., Fitzgerald, B., Hissam, S. A., and Lakhani, K. R., editors, *Perspectives on Free and Open Source Software*, pp. 23-46. MIT Press, Cambridge, MA.
GHOSH R., PRAKASH V.V. (2000), « The Orbiten Free Software Survey », *First Monday,* 5(7).
GODFREY M.W. and TU Q. (2000), « Evolution in Open Source software : a case study », in *Proceedings of the International Conference on Software Maintenance (ICSM 2000)*, San Jose, California, pp. 131-142.
GONZALEZ-BARAHONA J.M., ROBLES G., ORTUNO PEREZ M., RODERO-MERINO L., CENTENO-GONZALEZ J., MATELLAN-OLIVERA V., CASTRO-BARBERO E., DE-LAS HERAS-QUIROS P. (2004), « Analyzing the anatomy of GNU/Linux distributions : methodology and case studies (Red Hat and Debian) », in : Koch, S. (Ed.), *Free/Open Source Software Development.* Idea Group Publishing, Hershey, Pa., pp. 27-58.
GREWAL R., LILIEN G.L. and MALLAPRAGADA G. (2006), « Location, Location, Location : How Network Embeddedness Affects Project Success in Open Source Systems », *Management Science*, 52(7), pp. 1043-1056.
HARS A. and OU S. (2001), « Working for free ? Motivations for participating in Open Source projects », in *Proceedings of the 34th Hawaii International Conference on System Sciences,* Hawaii.
HEMETSBERGER A. and REINHARDT C. (2009), « Collective Development in Open-Source Communities : An Activity Theoretical Perspective on Successful Online Collaboration », *Organization Studies,* 30(9), pp. 987-1008.
HENKEL J. (2006), « Selective Revealing in Open Innovation Processes : The Case of Embedded Linux », *Research Policy,* 35, pp. 953-969.
HERTEL G., NIEDNER S., HERMANN S. (2003), « Motivation of software developers in open source projects : An internet-based survey of contributors to the Linux kernel », *Research Policy,* 32(7), pp. 1159-1177.
HOLCK J. and JORGENSEN N. (2004), « Do not check in on red : Control meets anarchy in two open source projects », in Koch, S., editor, *Free/Open Source Software Development,* pp. 1-26. Idea Group Publishing, Hershey, PA.
HOWISON J., CROWSTON K. (2004), « The perils and pitfalls of mining SourceForge », in : *Proc. of the International Workshop on Mining Software Repositories.* Edinburgh, Scotland, pp. 7-11.
HUNT F., JOHNSON P. (2002), « On the pareto distribution of sourceforge projects », in : *Proc. Open Source Software Development Workshop,* Newcastle, UK, pp. 122-129.
JORGENSEN N. (2001), « Putting it all in the trunk : Incremental software engineering in the FreeBSD Open Source project », *Information Systems Journal,* 11(4), pp. 321-336.
KEMERER C.F. (1987), « An Empirical Validation of Software Cost Estimation Models », *CACM*, 30(5), pp. 416-429.
KEMERER C.F. and SLAUGHTER S. (1999), « An Empirical Approach to Studying Software Evolution », *IEEE Transactions on Software Engineering,* 25(4), pp. 493-509.
KITCHENHAM B. and MENDES E. (2004), « Software Productivity Measurement Using Multiple Size Measures », *IEEE Transactions on Software Engineering,* 30(12), 1023-1035.
KOCH S. (2004), « Profiling an open source project ecology and its programmers », *Electronic Markets,* 14(2), pp. 77-88.
Koch, S. (2007), « Software Evolution in Open Source Projects – A Large-Scale Investigation », *Journal of Software Maintenance and Evolution,* 19(6), pp. 361-382.
KOCH S. (2008), « Effort Modeling and Programmer Participation in Open Source Software Projects », *Information Economics and Policy,* 20(4), pp. 345-355.

KOCH S. (2008a), « Measuring the Efficiency of Free and Open Source Software Projects Using Data Envelopment Analysis », in Sowe, S.K., Stamelos, I., and Samoladas, I. (eds.) : *Emerging Free and Open Source Software Practices,* pp. 25-44, Hershey, PA : IGI Publishing.

KOCH S. (2009), « Exploring the Effects of SourceForge.net Coordination and Communication Tools on the Efficiency of Open Source Projects using Data Envelopment Analysis », *Empirical Software Engineering,* 14(4), pp. 397-417.

KOCH S. and NEUMANN C. (2008), « Exploring the Effects of Process Characteristics on Product Quality in Open Source Software Development », *Journal of Database Management,* 19(2), pp. 31-57.

KOCH S., SCHNEIDER G. (2002), « Effort, Cooperation and Coordination in an Open Source Software Project : Gnome », *Information Systems Journal,* 12(1), pp. 27-42.

KRISHNAMURTHY S. (2002), « Cave or community ? An empirical investigation of 100 mature open source projects », *First Monday,* 7(6).

KUK G. (2006), « Strategic Interaction and Knowledge Sharing in the KDE Developer Mailing List », *Management Science,* 52(7), pp. 1031-1042.

LAKHANI K. R. and WOLF R. G. (2005), « Why hackers do what they do : understanding motivation and effort in free/open source software projects », in Feller, J., Fitzgerald, B., Hissam, S. A., and Lakhani, K. R., editors, *Perspectives on Free and Open Source Software,* pp. 3-22. MIT Press, Cambridge, MA.

LEHMAN M.M. and RAMIL J.F. (2001), « Rules and Tools for Software Evolution Planning and Management », *Annals of Software Engineering,* 11, pp. 15-44.

LONDEIX B. (1987), *Cost Estimation for Software Development,* Addison-Wesley, Wokingham, UK.

LONG Y. and SIAU K. (2007), « Social Network Structures in Open Source Software Development Teams », *Journal of Database Management,* 18(2), pp. 25-40.

MACCORMACK A., RUSNAK J. and BALDWIN C.Y. (2006), « Exploring the Structure of Complex Software Designs : An Empirical Study of Open Source and Proprietary Code », *Management Science,* 52(7), pp. 1015-1030.

MATSON J.E., BARRETT B.E., MELLICHAMP J.M. (1994), « Software Development Cost Estimation Using Function Points », *IEEE Transactions on Software Engineering,* 20(4), pp. 275-287.

MCCABE T. (1976), « A complexity measure », *IEEE Transactions on Software Engineering,* 2(4), pp. 308-320.

MCCONNELL S. (1999), « Open-source methodology : Ready for prime time ?», *IEEE Software,* 16(4), pp. 6-8.

MICHLMAYR M. (2005), « Software Process Maturity and the Success of Free Software Projects », in Zielinski, K. and Szmuc, T. (eds.) : *Software Engineering : Evolution and Emerging Technologies,* pp. 3-14, Amsterdam, The Netherlands : IOS Press.

MILLS H.D. (1971), *Chief Programmer Teams : Principles and Procedures.* Report FSC 71-5108. IBM Federal Systems Division.

MOCKUS A., FIELDING R., HERBSLEB J. (2002), « Two case studies of open source software development : Apache and Mozilla », *ACM Transactions on Software Engineering and Methodology,* 11(3), pp. 309-346.

NORDEN P.V. (1960), « On the anatomy of development projects », *IRE Transactions on Engineering Management,* 7(1), pp. 34-42.

OH W. and JEAN S. (2007), « Membership Herding and Network Stability in the Open Source Community : The Ising Perspective », *Management Science,* 53(7), pp. 1086-1101.

O'MAHONY S. and BECHKY B.A. (2008), « Boundary Organizations : Enabling Collaboration among Unexpected Allies », *Administrative Science Quarterly,* 53, pp. 422-459.

O'MAHONY S. and FERRARO F. (2007), « The Emergence of Governance in an Open Source Community », *Academy of Management Journal,* 50(5), pp. 1079-1106.

PAULSON J. W., SUCCI G. and EBERLEIN A. (2004), « An empirical study of open-source and closed-source software products », *IEEE Transactions on Software Engineering,* 30(4), pp. 246-256.

PERENS B. (1999), « The Open Source Definition », in DiBona, C. *et al.* (eds.), *Open Sources : Voices from the Open Source Revolution,* Cambridge, Massachusetts : O'Reilly & Associates.

PUTNAM L.H. (1978), « A general empirical solution to the macro software sizing and estimating problem », *IEEE Transactions on Software Engineering,* 4(4), pp. 345-361.

RAYMOND E.S. (1999), *The Cathedral and the Bazaar.* Cambridge, Massachusetts : O'Reilly & Associates.

ROBBINS J. (2005), « Adopting Open Source Software Engineering (OSSE) Practices by Adopting OSSE Tools », in Feller, J., Fitzgerald, B., Hissam, S.A., and Lakhani, K.R. (eds), *Perspectives on Free and Open Source Software,* Cambridge, MA : MIT Press.

ROBERTS J.A., HANN I.-H. and SLAUGHTER S.A. (2006), « Understanding the Motivations, Participation, and Performance of Open Source Software Developers : A Longitudinal Study of the Apache Projects », *Management Science*, 52(7), pp. 984-999.

ROBLES G., GONZALEZ-BARAHONA J. M., CENTENO-GONZALEZ J., MATELLAN-OLIVERA V., and RODERO-MERINO L. (2003), « Studying the evolution of libre software projects using publicly available data », in *Proceedings of the 3rd Workshop on Open Source Software Engineering, 25th International Conference on Software Engineering,* pp. 111-115, Portland, Oregon.

ROBLES G., KOCH S. and GONZALEZ-BARAHONA J. M. (2004), « Remote analysis and measurement of libre software systems by means of the CVSanalY tool », in *ICSE 2004 - Proceedings of the Second International Workshop on Remote Analysis and Measurement of Software Systems (RAMSS '04)*, pp. 51-55, Edinburgh, Scotland.

ROBLES G., GONZALEZ-BARAHONA J.M. & MERELO J.J. (2006), « Beyond source code : the importance of other artifacts in software development (a case study) », *Journal of Systems and Software,* 79(9), 1233-1248.

ROBLES G., GONZÁLEZ-BARAHONA J. M., IZQUIERDO-CORTAZAR D., & HERRAIZ I. (2009), « Tools for the study of the usual data sources found in libre software projects », *International Journal of Open Source Software & Processes,* 1(1), pp. 24-45.

RUSOVAN S., LAWFORD M., and PARNAS D.L. (2005), « Open Source Software Development : Future or Fad ?», in Feller, J., Fitzgerald, B., Hissam, S.A., and Lakhani, K.R. (eds), *Perspectives on Free and Open Source Software,* Cambridge, MA : MIT Press.

SCACCHI W. (2004), « Understanding Free/Open Source Software Evolution », in Madhavji, N.H., Lehman, M.M., Ramil, J.F. & Perry, D. (eds.), *Software Evolution,* New York : John Wiley and Sons Inc.

SCACCHI W., FELLER J., FITZGERALD B., HISSAM S. and LAKHANI K. (2006), « Understanding Free/Open Source Software Development Processes », *Software Process : Improvement and Practice,* 11(2), pp. 95-105.

SOWE S. K., ANGELIS L., STAMELOS I., MANOLOPOULOS Y. (2007), « Using Repository of Repositories (RoRs) to Study the Growth of F/OSS Projects : A Meta-Analysis Research Approach », *OSS 2007*, pp. 147-160.

STALLMAN Richard M. (2002), *Free Software, Free Society : Selected Essays of Richard M. Stallman.* Boston, Massachusetts : GNU Press.

STAMELOS I., ANGELIS L., OIKONOMOU A. and BLERIS G.L. (2002), « Code quality analysis in open source software development », *Information Systems Journal,* 12, pp. 43-60.

STEWART K.J. (2004), « OSS Project Success : From Internal Dynamics to External Impact », in *Collaboration, Conflict and Control : The 4th Workshop on Open Source Software Engineering (ICSE 2004)*, Edinburgh, Scotland.

STEWART K.J., AMMETER A.P. and MARUPING L.M. (2006), « Impacts of Licence Choice and Organisational Sponsorship on User Interest and Development Activity in Open Source Software Projects », *Information Systems Research,* 17(2), pp. 126-144.

STEWART K. J. and GOSAIN S. (2006), « The Moderating Role of Development Stage in Affecting Free/Open Source Software Project Performance », *Software Process : Improvement and Practice,* 11(2), pp. 177-191.

SUBRAMINIAN C., SEN R. and NELSON M.L. (2009), « Determinants of open source software project success : A longitudinal study », *Decision Support Systems,* 46, pp. 576-585.

VIXIE P. (1999), « Software Engineering », in : DiBona, C., Ockman, S., Stone, M. (Eds.), *Open Sources : Voices from the Open Source Revolution.* O'Reilly & Associates, Cambridge, MA, pp. 91-100.
Von KROGH, Georg and SPAETH, Sebastian (2007), « The open source software phenomenon : characteristics that promote research », *Journal of Strategic Information Systems,* 16(3), pp. 236-253.
WEISS D. (2005), « Measuring Success of Open Source Projects Using Web Search Engines », in *Proceedings of the 1st International Conference on Open Source Systems,* pp. 93-99, Genoa, Italy.
WHEELER D.A. (2005), More Than a Gigabuck : Estimating GNU/Linux's Size - Version 1.07 (updated 2002). Retrieved May 21, 2010, from http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html
YU L. (2006), « Indirectly predicting the maintenance effort of open-source software », *Journal of Software Maintenance and Evolution,* 18(3), pp. 311-332.
ZHAO L., ELBAUM S. (2000), « A survey on quality related activities in open source », *Software Engineering Notes,* 25(3), pp. 54-57.