

# A Look Inside the Forge:

## Developer Productivity and Spillovers in Open Source Projects

J. Eilhard<sup>1</sup> & Y. Ménière<sup>2</sup>

### *Abstract*

This paper presents an empirical study on the production of open source software, based on a panel of 10,553 projects registered on SourceForge over a period of 28 months (February 2005 until May 2007). We use a flexible Translog specification to estimate a production function relating the number of program updates with the number of corporate and voluntary contributors, taking into account the spillovers flowing from other projects. We find that corporate developers are the more productive ones, but that associating them with other developers in a project entails inefficiencies. We also find evidence of non-decreasing returns to scale in open source projects, thus suggesting substantial efficiency gains of division of labor in large open source projects. Our empirical analysis finally highlights a significant impact of spillovers on productivity. Spillovers mainly benefit mature projects. They especially flow between projects with the same topic and/or programming language, and have a stronger impact in projects involving corporate developers.

---

<sup>1</sup>CERNA, Mines Paristech, Paris

<sup>2</sup>CERNA, Mines Paristech, Paris

## *Introduction*

“The ability of the [open source software] process to collect and harness the collective IQ of thousands of individuals across the Internet is simply amazing.” This comment stems from a Microsoft software engineer made in the so-called Halloween memos in 1998 (Valloppillil, 1998). Already ten years ago, the world’s largest software producer understood the potential of open source software production. Today, this original productive model has demonstrated economic success. As a matter of fact, major proprietary applications face serious competition from open source rivals such as Linux or Apache. Even more significantly, the open source movement has largely permeated corporate software development. Google’s Android mobile phone platform, Sun’s OpenOffice office suite or, to a lesser extent, Apple’s App Store bear witness to the phenomenal power of the collaborative effort of thousands of individuals.

The success of open source software is puzzling in many respects. Open source licences require that source code be disclosed and available to all for free. Some so called restrictive licences, such as the General Public License, even impose that any extension of open source software be in turn licensed under the same terms. These legal provisions confer open source software the characteristics of a public good, and make it all the more striking that a large number of developers contribute to develop it. Against this backdrop, research in economics has been mostly interested in highlighting the various reasons why firms and private developers do contribute (Lerner & Tirole, 2002; Bonaccorsi & Rossi, 2003a; Lakhani & Wolf, 2005).

In this paper, we explore another key aspect of the success of open source, namely the process of software code production in open source projects. Using a panel of 10,553 open source projects tracked on the SourceForge repository website from February 2005 until June 2007, we estimate the individual productivity of voluntary and corporate developers, as well as spillovers flowing from other projects. Our choice of a flexible translog specification of the project production function makes it possible to assess scale effects and interactions between voluntary and corporate developers.

Our empirical analysis sheds light on several important features of open source production. We find that corporate developers are more productive than voluntary ones. However a negative interaction term indicates that associating the two groups entails a decrease in global productivity. Our results also suggest that the way open source projects are organized makes it possible to manage large projects quite efficiently. Indeed the production of open source code is not subject to strong decreasing returns to scale, thereby contrasting with proprietary software production.

Finally, we provide evidence of knowledge spillovers flowing between projects of the SourceForge repository. The possibility to reuse code from other projects is frequently presented as a strong advantage of the open source model (Raymond, 2001). Our findings confirm this statement: the amount of software code available in other projects substantially increases a project’s productivity. Spillovers benefit mainly to mature projects, and take place between projects that share the same topic and/or programming language. Moreover, it appears that corporate programmers are better able to exploit spillovers than voluntary ones.

As far as we know, this paper is one of the first empirical studies on the open source production process. Three recent papers, all of which using data from the SourceForge repository, investigate related topics like sorting of programmers and R&D spillovers. Belen-zon & Schankerman (2008) use the license types to infer the motives of programmers. They find in particular that corporate sponsorship – reported through a separate survey – tends to attract more contributions by “non-restrictive” developers, and does not affect contributions by “restrictive” developers. Using a different method (based on the email addresses of developers) to track corporate involvement, we complement their work by finding evidence of a productivity loss when corporate developers interact with voluntary ones. Using the number of downloads as a measure of outputs, Fershtman & Gandal (2008) show in turn that substantial spillovers flow within networks of projects and networks of developers. In a similar vein, Singh et al. (2008) focus on spillovers due to personal relationships among devel-

opers. Relating source code contributions with repeated interactions, they argue that trust and cohesion determine the size of the knowledge spillovers and thus affect the success of a project. By contrast, we propose another approach of spillovers, which is directly based on the availability of source code. Using source code releases as a measure of output, we take the quantity of code available in other projects as the source of spillover flows. We can thereby relate these flows with the projects' development stage, topic, programming language, and category of programmers.

The article is structured in seven Sections. We review in Section 2 the economic literature on open source production. Next, we develop our regression model (Section 3) and, in turn, present our data (Section 4). We discuss our results in Sections 5 and 6, the latter being specifically devoted to spillovers. Section 7 concludes.

### *Literature review*

This section reviews the literature on open source software production. After quickly presenting the incentives of developers, we discuss the organization of code production within projects and the effects of knowledge spillovers in open source software.

### **Programmers**

Traditionally, open source developers have been volunteers who contribute code for individual motives. Studies find that between 59% (Lakhani & Wolf, 2005) and 49% (Ghosh et al., 2002) of surveyed open source contributors are unpaid volunteers. They submit pieces of source code for a variety of reasons. Lakhani & Wolf (2005) find that 59% of their respondents contribute because they need a specific feature. Some also contribute because it is fun to solve challenging programming tasks (Shah, 2006), for ideological motives opposing free software to proprietary software, or to improve or signal their programming skills (Lerner & Tirole, 2002).

Apart from private volunteers, firms increasingly participate in open source development as well. According to a recent survey, they develop 15% of all open source software world-

wide (UNU-MERIT, 2006). More strikingly, Lakhani & Wolf (2005) find that around 40% of surveyed open source developers are paid to contribute to projects. There are different reasons why firms use open source software. It may be a cheaper alternative to in-house software development or buying proprietary software (Eilhard, 2008). Firms can also make profits by combining it with the proprietary code, hardware or services they supply to consumers (Lerner & Tirole, 2005; Henkel, 2006). Why firms contribute code is a more difficult question. Available surveys highlight various motives. Besides complying with open source licenses, these motives especially include being accepted by the community of developers and influencing the development path of the software (Henkel, 2006).

It is unclear whether corporate programmers are more productive than voluntary ones. Empirical evidence shows that voluntary and corporate contributors spend approximately the same amount of time on working in open source projects (Ghosh et al., 2002), although we do not know exactly whether they allocate their time equally on the same types of projects. Differences in individual productivity may also depend on skills. Corporate programmers are paid professionals, while many voluntary programmers contribute to projects to demonstrate or improve their skills (UNU-MERIT, 2006).

## **Organization**

The most striking feature of open source production being that it is realized by communities of peers (von Hippel & von Krogh, 2003), several authors seek to understand how coordination works within each project. Lerner & Tirole (2002) emphasize the existence of informal hierarchies in open source projects. Each project is organized in circles of more or less involved developers (from those who report bugs to those who fix them) around one or several ‘charismatic’ leaders recognized as such by their peers. By contrast, Benkler (2002) contends that the organization of open source production differs from both the hierarchical and market models. In his view, labour division between small contributors hinges on the combination of voluntary contribution and modular software design, which the development

of information and communication technologies has made possible.

Arguably, this productive model can ensure a more efficient division of labor than traditional ways of developing software code (Bonaccorsi & Rossi, 2003c)<sup>3</sup>. The modular design makes it possible for volunteers to select for themselves on which problems they want to work, without having to care about the other modules. Matching their skills with the tasks at hand, they moreover take up tasks which more closely fit their set of capabilities (Amabile, 1983). Self-assignment thus can lead to a better use of human capital in open source projects and hence to a higher effectiveness compared to traditional software production (von Krogh et al., 2003).

In a recent empirical study based on SourceForge projects, Belenzon & Schankerman (2008) find that the sorting of programmers also strongly depends on the license type, project's size and corporate sponsorship. Interestingly, they show that the effect of the license type on contributions varies with the project type: a restrictive license will draw more contributions for end user projects, and less for developer tool projects.

### **Firms' involvement in open source projects**

Cooperation between voluntary and corporate developers is of particular interest. The open innovation paradigm (Chesbrough, 2003) suggests that complementarities in skills and objectives may further boost open source production. Firms are for instance more interested in improving user interfaces or interoperability with hardware – two dimensions which are usually neglected by voluntary user-developers (Lerner & Tirole, 2002; Scotchmer & Maurer, 2006). Yet corporate developers may also be reluctant to share source code if the competitive advantage of their firm is at stake (Henkel, 2008). As a result, they practice various forms of selective revealing to protect valuable information from leaking to open source projects (Bonaccorsi & Rossi, 2003b; Dahlander & Wallin, 2006; Fosturi et al., 2008).

---

<sup>3</sup>As Brooks (1978) notes in his famous phrase that “adding manpower to a late project makes it later”, traditional software production quickly runs into decreasing returns to scale. Brooks (1978) attributes rising coordination cost as the reason for decreasing productivity. In traditional software creation, changes in one part of the software have ramifications on all other parts of the software. Accordingly, each developer needs to know what all other developers do. As a consequence, the cost of communication increases exponentially as more developers join the project.

Reflecting the conflict of interest, Henkel (2006) observes that 51% of surveyed firms do not reveal their contributions to open source projects at all. Still, many firms contribute to open source projects, some of them intensively (Dahlander & Magnusson, 2005). Beyond compliance with open source licenses, they do so to be able to influence the project's development path, which in turn requires being perceived as a good player by voluntary programmers (Henkel, 2006). Accordingly, corporate developers tend to work in key positions in open source projects, and to interact more frequently than others with core developers (Dahlander & Wallin, 2006).

## **Spillovers**

Spillovers are another aspect of the open innovation paradigm. As stated by Raymond (2001): "Good programmers know what to write. Great ones know what to rewrite (and reuse)." Indeed source code under open source licenses being publicly available, the stock of open source software has become a wide library in which programmers are free to pick and reuse the pieces of code they need for a particular project. Put in economic terms, the knowledge – the code – generated for each projects spills over to the other projects.

Although spillovers may be a key factor in the spectacular growth of the open source movement, empirical studies have been missing until very recently. Fershtman & Gandal (2008) focus on spillovers flowing through projects involving the same developers, and through developers involved in the same project. They show that a project indeed benefits of spillovers when (i) it is directly connected to other projects (ii) it is close to a large number of projects beyond their immediate connections (iii) it occupies a central position in a network of projects. Taking a closer look at contributors, Singh et al. (2008) explore spillovers which accrue through social interaction. They argue that projects are more successful when developers have (i) a high degree of internal cohesion, which leads to mutual trust and reciprocity norms, and (ii) a moderate level of relationships with developers outside the project. This suggests that, while information flows among developers are important, spillovers are not

restricted to projects that share common developers.

### *Specification of the Econometric Model*

The measurement of open source projects' productivity encounters the same type of difficulties as any attempt to measure the true productivity of research (Hausman et al., 1984). Ideally, we would need a measure of the social return of each project, including the utility users derive from the software as well as the spillovers to other projects. Unlike corporate research, the prices of innovative goods cannot proxy spillovers, let alone users' utility, for open source software is distributed royalty free. It might be more realistic to infer private returns, namely the returns for contributors, from their degree of individual involvement. Such a work would require precise information on their individual productivity and the time they spend on each project.

Our objective in this paper is more modest and constitutes a first step in that direction. We focus on the determinants of technical success in open source software development, as measured by software releases. Such releases signal that a new version of the open source software is ready for download. They provide an interesting proxy of innovative output since released versions are generally improvements upon the previous versions which they replace.

We have no priors about the true functional form of the open source production function. Since our dependent variable – the counts of new releases – only takes on non-negative integer values, we assume that it is generated by a Poisson process. We model the single parameter of the Poisson distribution function,  $\lambda$ , as a function  $F(X, \beta)$  where  $X$  is a vector of explanatory variables and  $\beta$  is a vector of parameters and the error term  $\epsilon$ .

$$E(Y_i) = \lambda_i = e^{F(\cdot)\epsilon}$$

We want to analyze the productivity of the three different groups of contributors. Without any prior constraints on the function  $F(X_{it}, \beta)$ , we estimate the coefficients of a translog production function for all open source projects. The translog function is an attractive

flexible specification. It has both linear and quadratic terms with the ability of using more than two factor inputs, and can be approximated by a second order Taylor series (Christensen & Greene, 1976). Applied to our three labor inputs, the translog production function can be written in terms of logarithms as:

$$F(N_k, S, P) = \beta_A \ln N_A + \beta_C \ln N_C + \beta_P \ln N_P + \frac{1}{2} \beta_{AA} \ln N_A^2 + \frac{1}{2} \beta_{CC} \ln N_C^2 + \frac{1}{2} \beta_{PP} \ln N_P^2 + \beta_{AC} \ln N_A \ln N_C + \beta_{PA} \ln N_P \ln N_A + \beta_{CP} \ln N_C \ln N_P + \gamma_1 S_i + P_i \delta$$

$N_A, N_C$  and  $N_P$  are the respective numbers of developers for academic, corporate or private developer groups,  $S_i$  is a vector of spillover proxies and  $P_i$  is a vector of control variables, including dummies for each development stage, the project's topic, as well as for each month.

Observe that the translog function can be transformed in a standard Cobb-Douglas function by imposing zero-value coefficients on the second order terms. This would imply that the estimated elasticity of output with respect to each input be constant by assumption. For sake of comparison, we also estimate this constrained specification.

Besides its flexibility, another advantage of the translog function lies in the interpretation of the second order terms. The sign of the coefficients for the squared logarithms of labor inputs hints towards increasing or decreasing elasticity of output. We can therefore use it as an indicator of increasing or decreasing returns with respect to each category of programmers. Moreover, positive or negative coefficients for the crossed logarithms make it possible to derive conclusions on the effect of interactions between different categories of developers.

Estimating a production function creates a simultaneity problem (Marschak & Andrews, 1944; Olley & Pakes, 1996). In the traditional setting, firms choose the factors of production according to profit-maximizing, first-order conditions. Because they consider firm-specific productivity differences, more productive firms use more factors of production, rendering it more difficult to establish a causal link between production factors and output. The control variables in  $P_i$  are a first way to capture differences in productivity between projects. These variables include dummies for the development stage of the projects and their topic of

application, and several channels of knowledge spillovers from other projects. We moreover address the simultaneity problem by using a fixed-effect estimation method. This approach eliminates major misspecification, which are transmitted to the factor decisions and are constant over time (Griliches & Mairesse, 1998). It however requires us to assume that unanticipated elements of the error term at period  $t$  do not affect factor decisions at later periods.

## *Data*

We use a balanced panel of 10,553 open source projects tracked on SourceForge from February 2005 until June 2007 ( $T = 28$  months). SourceForge is an internet platform for open source projects. It helps new projects attract developers and users. Sourceforge provides the necessary tools for managing a software project, such as user fora, bandwidth for downloads and a version control system to keep track of contributions. SourceForge is the largest internet repository with almost 300,000 projects, similar, but considerably smaller, platforms are GoogleCode or Kenai.

SourceForge as an incubator includes a large number of small or inactive projects. Since creating new projects is costless, many projects may be started, just to be subsequently abandoned after a short while. For this reason Howison & Crowston (2004) caution the use of SourceForge projects to draw conclusions about open source projects in general. In this regard our approach is especially valuable, because we only look at projects which have already released a version of their software, thus avoiding inactive projects all together.

Open source projects regularly release newer versions of their software. These file releases fix programming bugs, add new features or improve performance. Larger projects generally announce their releases in advance; smaller projects may post updates intermittently. There are various forms in which users can obtain these releases: compressed source code, binary

installers or text files. To have comparable data, we only look at compressed files.<sup>4</sup> This allows us to consider projects across different operating systems, since binary installers often are operating system specific.

In addition to information on software projects, we obtained developer backgrounds using their email addresses. These email addresses were related to the corresponding websites and sorted into three categories: academic, private and corporate. We sampled ten websites to obtain keywords for each category and counted the occurrences of each word. We established a ranking and were thus able to attribute an email address with either category, for example 'hotmail.com' to the private category or 'ibm.com' to the corporate one. In all, we checked approximately 15,000 websites. We then retained projects where all developer backgrounds were known as well those which actually released a file update during the observed time period. so we could account for all registered developers in a project, 10,553 projects remained in our sample.

This method clearly has two drawbacks. On the one hand, we might see a type I error. We overestimate the number of corporate developers for those who use their work email for private purposes. They subscribe to SourceForge using their office email and thus pass as corporate developers, even though their company does not directly promote open source development. On the other hand, we might see a type II error. Contributors might use their private email addresses for subscription, although their firms do support open source projects. There is no direct way to mitigate these issues. One possibility is to argue by deduction. Finding statistically significant differences between corporate, private and academic developers might support the validity of our data collection method.

## **Descriptives**

SourceForge offers 222 different topics for open source software. After regrouping these categories in broader application fields, we obtain a more manageable 19 topics. Figure 1 shows the number of projects per topic. We immediately see that a lot of projects focus

---

<sup>4</sup>In fact, we only look at gzip files, as these were the most common types of compressed files in the data base.

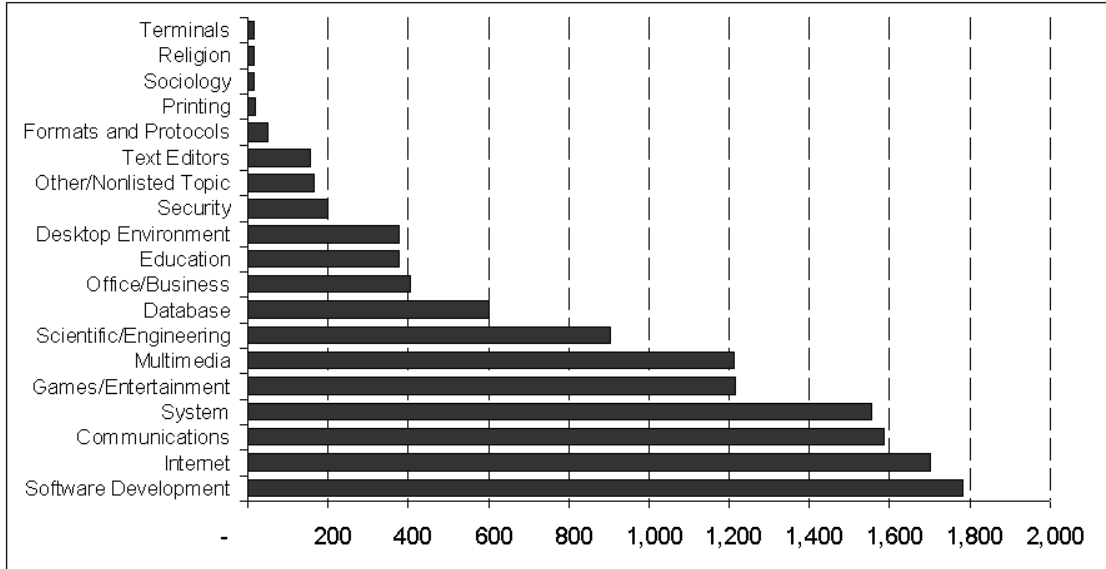


Figure 1: Projects per Topic

on four topics: software development, internet, communications and system administration. The figure also shows the tremendous variety in topics that we find on SourceForge. There are projects on religion - for instance an application to calculate the Islamic prayer times - as well as scientific software and games - incidentally, approximately 200 versions of Tetris are available on Sourceforge.

The number of developers is a possible indicator for project size. Figure 2 shows that our sample contains an overwhelmingly large number of small projects. We see that almost 80% of the observed projects have only one registered developer over the entire time period. This begs the question whether these project have a significantly different mode of production than projects with more developers. To test the impact of this possible bias, we will estimate the production function with a sub-sample of the projects with at least two developers ( $n' = 2,406$ ). A more detailed look at the developer categories reveals that on average each project has at least one private developer and every fifth project either a corporate or an academic contributor (Table 1).

Projects release new updates irregularly. Figure 3 depicts the number of projects for the maximal number of file releases within the 28 months. We see that the majority of projects

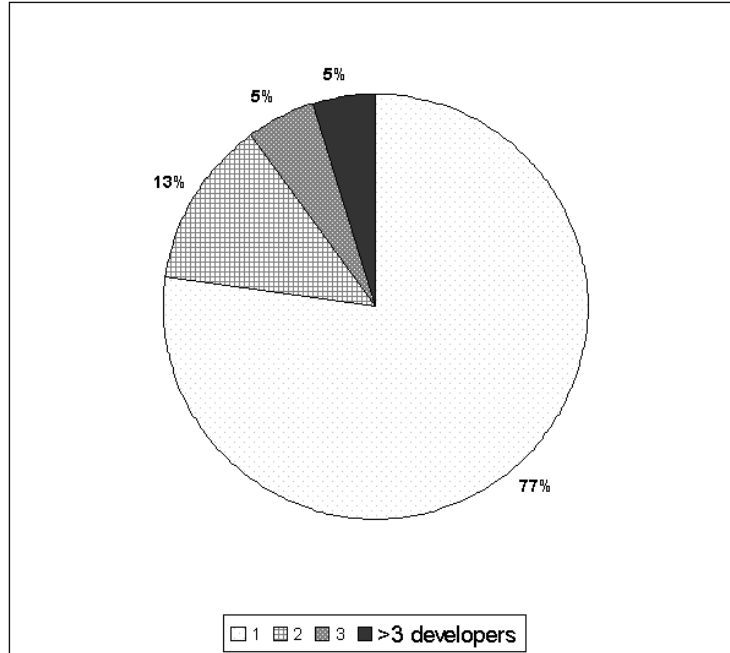


Figure 2: Maximal Number of Developers per Projects

Table 1: Summary Statistics

Variable	Mean	Median	Std. Dev.	Min	Max	Observations (n*T)
Academic Developers	0.200		0.650	0	12	295,484
Corporate Developers	0.223		0.587	0	12	295,484
Private Developers	1		0.827	0	15	295,484
File Releases	1.347		1.198	0	21	295,484
Spillovers - same Topic (Tbytes)	80.5	22.0	163.3	0	611.1	295,484
Spillovers - same Topic/same Programming Language (Tbytes)	9.9	1.1	35.7	0	410.0	295,484
Spillovers - same Topic/diff. Programming Language (Tbytes)	70.6	17.1	148.5	0	611.1	295,484
Spillovers - different Topic/same Programming Language (Tbytes)	85.03	51.1	118.84	0	515.76	295,484

has only one release over the entire period and that there are few projects which are very productive, having more than ten releases.

We measure spillovers with the number of terbytes all projects with the same topic. Table 1 shows that average size of the code commons is considerable. Projects have on average 81 terabytes of - compressed - source code available to copy and reuse. The refined knowledge base of projects with same topic and same programming language still encompasses almost 10 terabytes of source code which is about the size of the printed collection of the U.S. Library of Congress.

SourceForge provides a way to measure the development status of each project. Project administrators indicate in which of the seven different stages the project is: planning, pre-

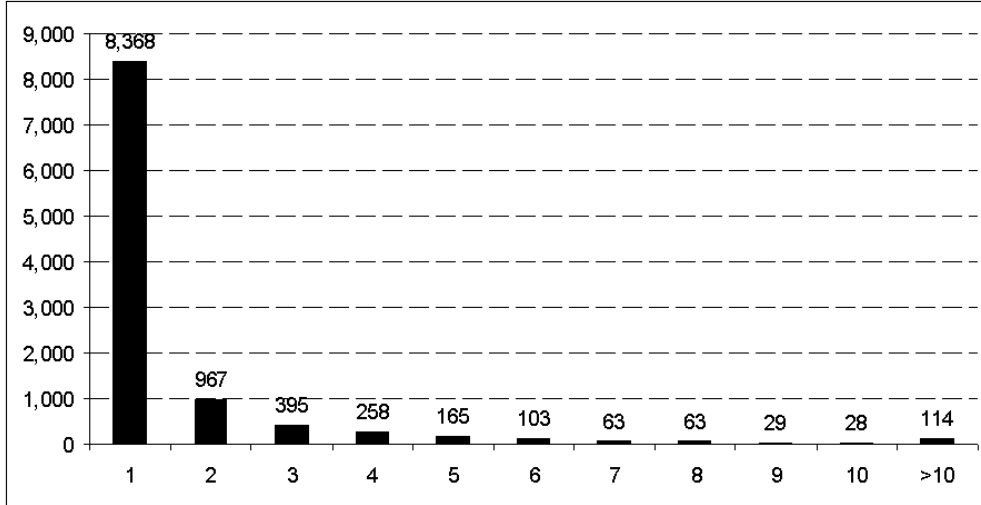


Figure 3: Number of Releases

Table 2: Development Stages

Variable	Mean	Std. Dev.	Min	Max	Observations
Planning	0.04	0.18	0	1	295,484
Pre-Alpha	0.12	0.32	0	1	295,484
Alpha	0.20	0.40	0	1	295,484
Beta	0.31	0.46	0	1	295,484
Production	0.28	0.45	0	1	295,484
Mature	0.02	0.14	0	1	295,484
Inactive	0.00	0.06	0	1	295,484

alpha, alpha, beta, stable, mature or inactive. Although the choice of one stage over another is based on subjective criteria, it gives us a general idea of the progress and stability of the project. Table 2 shows the frequencies of projects within each development stage. The mean gives the percentage share of each category within our sample. We see that around 60% of the projects in our sample are in beta or later stages. This provides further evidence that we are indeed dealing with actively developed projects.

### *Empirical Results*

Table 5 shows the results of the regressions. Models 1 and 2 show the estimates for the full sample ( $n= 10,553$ ). We progressively render the functional form more flexible, beginning with a classic Cobb-Douglas production function (model 1) and using the complete translog specification (model 2). The same translog specification is estimated in models 9 and 10,

Table 3: Likelihood Ratio Test

Model 1 (Cobb-Douglas)	$\chi^2 = 224.31$	(0.000)
------------------------	-------------------	---------

respectively for the sample of projects with at least two developers ( $n' = 2,406$ ) and for projects in beta or later development stages. All regressions are fixed-effects models. A Hausman test shows that random-effects estimates are inconsistent, refuting the assumption that unobservable project characteristics are uncorrelated with past, present or future values of the regressors. Considering the nature of the independent variable, the count of file releases, the use of a Poisson panel regression lends itself rather directly. The absence of zeroes in the dependent variable prevents the use of negative binomial regressions to test for overdispersion. We argue instead by visual inspection (see table 1) that the assumptions for a Poisson regression are met, namely that  $Mean(Releases) = Var(Releases)$ .

We check the validity of our specification with a likelihood ratio test. Comparing the complete translog function (model 2) with a Cobb-Douglas specification (model 1), we test the null hypothesis that the each restricted model is a more adequate representation of the data than the translog function. Table 3 shows the chi-squared and the respective p-values in parentheses. No null hypothesis can be supported, thus suggesting that indeed model 2 is the best specification for our data.

### Developer Productivities

As can be seen in Table 5, we find significant coefficients for each category of developer in the Cobb-Douglas specification (model 1). Recall that by construction this specification implies constant elasticity of output with respect to each input. The estimated input coefficients measure these elasticities. They suggest a slightly lower elasticity for corporate programmers than for private and academic ones. However, this conclusion is not confirmed statistically, the coefficients being equal with a high probability (.8484).

Model 2 makes it possible to go further in the comparison. The translog function relaxes the assumption of constant elasticity by taking into account additional input variables,

Table 4: Average Computed Marginal Products (t-values)

<b>Marginal Product</b>	<b>Mean</b>	<b>Median</b>	<b>Std. Dev.</b>	<b>Corporate (t-values)</b>	<b>Private (t-values)</b>
Corporate Developers	0.51	0.43	0.461		
Private Developers	0.36	0.28	0.321	496.09***	
Academic Developers	0.29	0.25	0.284	496.09***	312.12***

namely the squared logs and the crossed logs of developers. This flexible specification also makes it possible to disentangle the different effects captured in each Cobb Douglas coefficient. Although the simple logs of developers remain significant in model 2, they have lower values than in model 1, and the estimated coefficient become now higher for corporate developers. Moreover, significant and positive coefficients for squared logs show that elasticity of output with respect to each category of developer is in fact increasing. In line with simple logs, these coefficients are higher for corporate developers than for the other two categories. As shown in Table 5, the crossed effects are significant only when corporate developers are involved, and then negative in each case. In other term, we find empirical evidence of a negative effect of interactions between corporate developers and other categories on the productivity of open source projects.

In sum, the productivity of corporate developers is subject to conflicting effects. On the one hand, coefficients of simple logs and squared logs suggest that corporate developers are more productive than other categories. On the other hand, corporate developers also have a negative effect on production when they are associated with other categories.

Making comparisons between categories of developers requires taking into account these conflicting effects. As a first step in this direction, we calculate the marginal product of each developer group for each project in model 2. Table 4 presents the average and the median computed marginal products for each developer group as well as the respective t-statistics for equality of means. We see that the average corporate marginal product is significantly higher than the marginal product for the other two groups. These results refute the hypothesis that corporate developers are less productive than other developers from other groups (hypothesis 1a). Corporate developers are on average more productive than private or

academic contributors. The results suggest that an additional corporate developer produces approximately 0.5 file releases more on average over 28 months.

### Scale effects

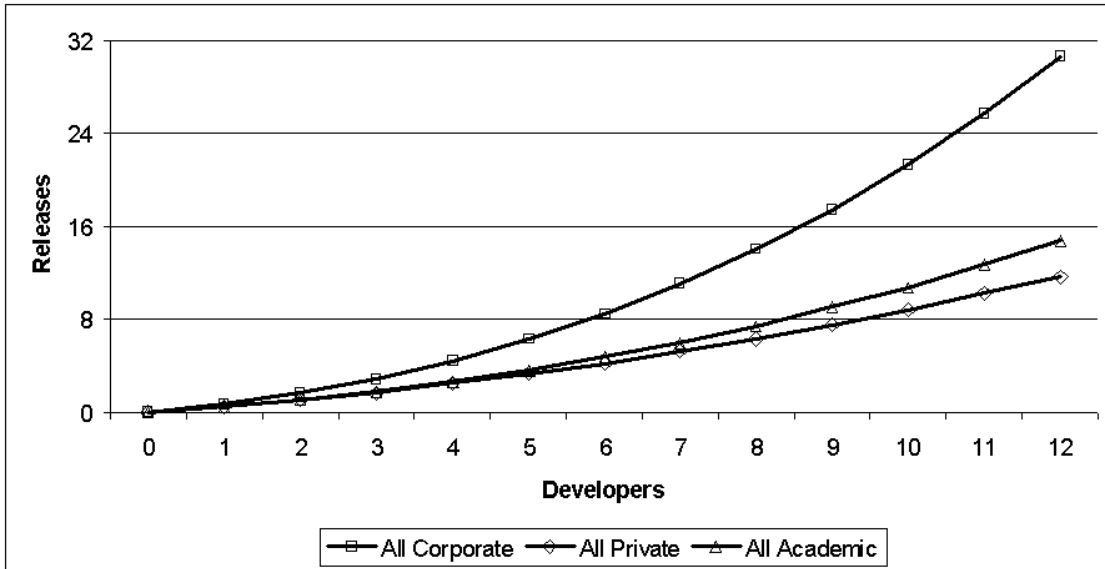
Looking at the average marginal product gives only a partial picture, for this leaves out diminishing or rising marginal productivity. To address this limitation, we consider now the relationship between productivity and the scale of open source projects.

The simple Cobb-Douglas specification in model 1 makes it possible to derive clear conclusions on returns to scale. The sum of the estimated coefficients for each category of programmers is equal to 1.9813 – much above 1 – which suggests that open source production is subject to increasing returns to scale. The quadratic and interaction terms make it more difficult to identify returns to scale in the translog model. To address this problem, we simulate five scenarios based on model 2: Projects with (i) all corporate, (ii) all private, (iii) all academic as well as projects with cooperation in which the number of contributors is (iv) equally divided between academic, private and corporate developers and (v) weighted according to their distribution in the overall sample. Figure 4 presents these simulations. In the first three scenarios the project’s output is clearly a convex function of the number of developers within the range of developers we observe. In other terms, each new contributor increases the average developer productivity. This is less obvious in the last two scenarios, where the effect of the interaction term is stronger. We nevertheless see a slight positive slope, suggesting constant or very slightly decreasing returns to scale.

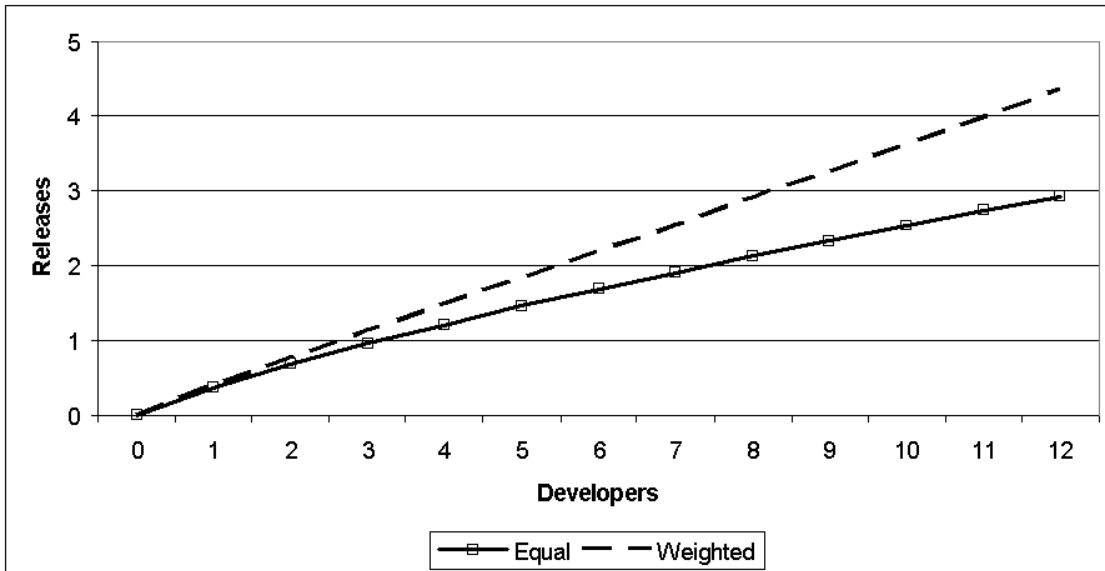
A possible factor driving these results could have been a qualitative difference in nature between the releases observed in nascent and more mature project. Developing the first version of a software product may indeed take more time than subsequent improvement. We have therefore estimated the same translog specification respectively for the sample of projects with at least two developers ( $n' = 2,406$ ) and for projects in beta or later development stages. As can be seen in Table 5 (models 9 and 10), the results do not differ substantially from model 2. Another possible explanation lies in the way we measure the developers’ contribution. The

Figure 4: Simulations

(a) Single Developer Groups



(b) Cooperation



increasing returns to scale we observe might reflect the fact that developers tend to devote more time on large mature projects – which we cannot control for. Still, our results indicate that open source production is hardly subject to decreasing returns. Despite a possible bias in time measurement, this sheds light on the intrinsic efficiency of this software production model. The combination of modular design and voluntary contributions is an efficient way to divide labor, even for large projects.

Table 5: Main Results

Releases	Model 2	Model 1	Model 10	Model 9
Corporate Developers (ln)	0.5477***	0.6259***	0.5240***	0.5252***
Private Developers (ln)	0.4033***	0.6675***	0.4296***	0.3726***
Academic Developers (ln)	0.3277*	0.6679***	0.1547	0.278
Corporate Developers (ln) <sup>2</sup>	0.3114***		0.2991***	0.2738***
Private Developers (ln) <sup>2</sup>	0.2292***		0.1951***	0.1999***
Academic Developers (ln) <sup>2</sup>	0.2918***		0.3749***	0.2760***
Corporate/Private Developers	-0.2569**		-0.2637**	-0.2390**
Corporate/Academic Developers	-0.1994***		-0.2324***	-0.1849***
Private/Academic Developers	-0.0844		0.0139	-0.0764
Spillovers - same Topic	0.00014***	0.00014***	0.00014***	0.0001
Topic: Database	-0.0772**	-0.0763**	-0.1354***	-0.0257
Topic: Desktop	-0.0155	-0.0154	-0.0116	0.0176
Topic: Education	0.0613	0.061	0.1134**	-0.0781
Topic: Format	-0.0166	-0.0169	-0.0082	-0.0849
Topic: Games	0.0031	0.0033	-0.0154	-0.0024
Topic: Internet	-0.0550***	-0.0567***	-0.0393	-0.1211***
Topic: Multimedia	-0.0029	-0.0032	-0.0025	0.0032
Topic: Office	-0.0414	-0.0423	-0.0308	-0.0897*
Topic: Others	-0.0817	-0.0812	-0.0926	-0.1885*
Topic: Printer	0.0593	0.0598	0.0674	0.0093
Topic: Religion	-0.1457	-0.1457	-0.178	0.2153
Topic: Science	-0.1502***	-0.1483***	-0.1419***	-0.0732*
Topic: Security	0.0122	0.0119	0.0305	-0.1493***
Topic: Sociology	-0.0454	-0.0462	-0.123	-0.0898
Topic: Software Development	0.018	0.0164	0.0232	---
Topic: System	-0.0612**	-0.0588**	-0.0367	-0.0601
Topic: Terminal	0.1041	0.1047	0.1019	-0.0921
Topic: Text Editors	-0.0443	-0.0439	-0.045	-0.0596
Pre-Alpha Status	-0.0198	-0.0203	---	0.0103
Alpha Status	0.1301***	0.1284***	---	0.1089*
Beta Status	0.2779***	0.2773***	0.1156***	0.2337***
Production Status	0.5865***	0.5870***	0.4433***	0.4707***
Mature Status	0.7309***	0.7350***	0.5499***	0.4023***
Inactive Status	-20.074	-20.0747	-20.2538	-18.153
Time	sig.	sig.	sig.	sig.
Number of obs	295,484	295,484	191,017	67,368
Number of groups	10,553	10,553	7,018	2,406
Wald chi2(16)	9,191	9,149	8,329	3,879
Log likelihood	-297,089	-297,121	-197,029	-70,596

\*: p < 10%    \*\*:p < 5%    \*\*\*: p < 1%

Table 6: Impact of Average Input Factors

Total Effect	Labor Factors	Spillovers
1	0.975	0.025

## Spillovers

In Tables 7 and 8, we present several models assessing the spillovers effects flowing between projects through various channels. They show that the coefficients for spillovers are positive and significant. The size of the code commons has a differentiated effect on production depending on the developer group, topic, programming language and the development stage of the project.

Model 2 includes a unique spillover variable denoting the number of terabytes of source code available in other projects of the same topic. Albeit small, we find a positive and significant coefficient for spillovers. Calculating the impact of the average input factors and spillovers on production, we find that spillovers account for 2.5%, whereas developers make out 97.5% of the entire effect (table 6).

It is worthwhile to look more closely at the channels through which spillovers flow. Looking at the interaction terms of developer groups and spillovers (model 3), we find that the coefficients are significant for corporate and academic contributors. The results indicate that adding another terabyte of code commons increases the productivities for these two developer groups, but not for the private one. It thus seems that academic and corporate contributors copy and reuse more than private developers and hence benefit more from the available code commons. There is no significant difference between the effects for academic and corporate contributors ( $\text{Chi}^2 = 1.02$ ). Model 4 considers the impact of the license type on spillovers. We find that none of the coefficients are significant. Surprisingly, sharing the same license type does not increase spillovers. And conversely, heterogeneous license types does not seem to be an impediment to spillovers.

Model 5 and 8 show that the further advanced a project is in its development, the more it benefits from the available source code in other projects. Spillovers are significant only

Table 7: Results for Spillovers A

Releases	Model 5	Model 4	Model 3	Model 2
Corporate Developers (ln)	0.5588***	0.4481***	0.5793***	0.5477***
Private Developers (ln)	0.4060***	0.3009***	0.4000***	0.4033***
Academic Developers (ln)	0.3122*	0	0.3130*	0.3277*
Corporate Developers (ln) <sup>2</sup>	0.2883***	0.2214**	0.2487***	0.3114***
Private Developers (ln) <sup>2</sup>	0.2287***	0.2181***	0.2300***	0.2292***
Academic Developers (ln) <sup>2</sup>	0.2992***	0.2227**	0.2900***	0.2918***
Corporate/Private Developers	-0.2509**	-0.2	-0.2366**	-0.2569**
Corporate/Academic Developers	-0.1976***	-0.1198*	-0.2145***	-0.1994***
Private/Academic Developers	-0.0847	-0.0156	-0.0883	-0.0844
Spillovers - same Topic	-0.0001		0	0.00014***
Spillovers - same Topic / Corporate Developers			0.0003***	
Spillovers - same Topic / Private Developers			0	
Spillovers - same Topic / Academic Developers			0.0002**	
Spillovers - same Topic / same License		0.0002		
Spillovers - same Topic / different License		0.0002*		
Spillovers - same Topic/ Pre-Alpha status	-0.0002			
Spillovers - same Topic/ Alpha status	-0.0001			
Spillovers - same Topic/ Beta status	0.0002*			
Spillovers - same Topic/ Production status	0.0006***			
Spillovers - same Topic/ Mature status	0.0006***			
Spillovers - same Topic/ Inactive status	-0.0015			
Topic: Database	-0.0744**	-0.0502	-0.0720**	-0.0772**
Topic: Desktop	-0.0185	-0.0031	-0.0163	-0.0155
Topic: Education	0.0667*	0.0703*	0.0630*	0.0613
Topic: Format	-0.0257	0.0003	-0.0181	-0.0166
Topic: Games	0.0051	0.0102	0.0044	0.0031
Topic: Internet	-0.0495**	-0.0415*	-0.0521**	-0.0550***
Topic: Multimedia	-0.0009	0.0023	-0.0018	-0.0029
Topic: Office	-0.0462	-0.0319	-0.0368	-0.0414
Topic: Others	-0.0893*	-0.0629	-0.0816	-0.0817
Topic: Printer	0.0688	0.0604	0.0617	0.0593
Topic: Religion	-0.1321	-0.1341	-0.1492	-0.1457
Topic: Science	-0.1471***	-0.1318***	-0.1490***	-0.1502***
Topic: Security	0.016	0.0377	0.0165	0.0122
Topic: Sociology	-0.0489	-0.0371	-0.0461	-0.0454
Topic: Software Development	0.0234	0.0288	0.0198	0.018
Topic: System	-0.0606**	-0.0751	-0.0674**	-0.0612**
Topic: Terminal	0.0893	0.1039	0.0938	0.1041
Topic: Text Editors	-0.04	-0.0241	-0.0404	-0.0443
Pre-Alpha Status	0.0136	0.0199	-0.0196	-0.0198
Alpha Status	0.1572***	0.0819***	0.1299***	0.1301***
Beta Status	0.2838***	0.1993***	0.2781***	0.2779***
Production Status	0.5595***	0.4644***	0.5872***	0.5865***
Mature Status	0.6798***	0.5914***	0.7334***	0.7309***
Inactive Status	-19.9702	-19.3951	-20.0743	-20.074
Time	sig.		sig.	
Number of obs	295,484	273,583	295,484	295,484
Number of groups	10,553	10,531	10,553	10,553
Wald chi2(16)	9,343	6,060	9,205	9,191
Log likelihood	-297,008	-273,906	-297,079	-297,089

\*: p < 10%    \*\*:p < 5%    \*\*\*: p < 1%

Table 8: Results for Spillovers B

Releases	Model 8	Model 7	Model 6
Corporate Developers (ln)	0.5650***	0.5521***	0.5478***
Private Developers (ln)	0.4090***	0.4063***	0.4034***
Academic Developers (ln)	0.3116*	0.3281*	0.3276*
Corporate Developers (ln) <sup>2</sup>	0.2830***	0.3090***	0.3114***
Private Developers (ln) <sup>2</sup>	0.2271***	0.2281***	0.2292***
Academic Developers (ln) <sup>2</sup>	0.3002***	0.2919***	0.2918***
Corporate/Private Developers	-0.2584**	-0.2619**	-0.2569**
Corporate/Academic Developers	-0.1965***	-0.1996***	-0.1994***
Private/Academic Developers	-0.0854	-0.0849	-0.0844
Spillover - same Topic/same Programming Language	-0.00076*	0.00015**	0.00014**
Spillover - same Topic/ Programming Language / Pre-Alpha stage	0.00004		
Spillover - same Topic/ Programming Language / Alpha stage	0.0004724		
Spillover - same Topic/ Programming Language / Beta stage	0.0006354		
Spillover - same Topic/ Programming Language / Production stage	0.00138***		
Spillover - same Topic/ Programming Language / Mature stage	0.00187***		
Spillover - same Topic/ Programming Language / Inactive stage	0.0021		
Spillover - same Topic/different Programming Language	-0.000041	0.00015***	0.00014***
Spillover - same Topic/diff. Programming Language/ Pre-Alpha stage	-0.00018		
Spillover - same Topic/diff. Programming Language/ Alpha stage	-0.00015		
Spillover - same Topic/diff. Programming Language/ Beta stage	0.00017		
Spillover - same Topic/diff. Programming Language/ Production stage	0.00045***		
Spillover - same Topic/diff. Programming Language/ Mature stage	0.0004**		
Spillover - same Topic/diff. Programming Language/ Inactive stage	-0.0022		
Spillover - different Topic/same Programming Language	0.000049**	0.00004**	
Topic: Database	-0.0735**	-0.0773**	-0.0772**
Topic: Desktop	-0.0177	-0.0151	-0.0155
Topic: Education	0.0659*	0.0607	0.0614
Topic: Format	-0.0238	-0.0168	-0.0166
Topic: Games	0.0055	0.0036	0.003
Topic: Internet	-0.0490**	-0.0545**	-0.0550***
Topic: Multimedia	0.0008	-0.0028	-0.0029
Topic: Office	-0.0443	-0.0413	-0.0414
Topic: Others	-0.0878*	-0.0807	-0.0817
Topic: Printer	0.0709	0.0607	0.0593
Topic: Religion	-0.1331	-0.1466	-0.1457
Topic: Science	-0.1461***	-0.1498***	-0.1502***
Topic: Security	0.0163	0.0126	0.0121
Topic: Sociology	-0.0654	-0.049	-0.0454
Topic: Software Development	0.0241	0.0183	0.018
Topic: System	-0.0589**	-0.0612**	-0.0612**
Topic: Terminal	0.0934	0.1054	0.1041
Topic: Text Editors	-0.0396	-0.0442	-0.0443
Pre-Alpha Status	0.0115	-0.0204	-0.0198
Alpha Status	0.1557***	0.1293***	0.1301***
Beta Status	0.2822***	0.2762***	0.2779***
Production Status	0.5572***	0.5846***	0.5866***
Mature Status	0.6818***	0.7291***	0.7309***
Inactive Status	-19.9647	-20.0742	-20.0739
Time	sig.	sig.	sig.
Number of obs	295,484	295,484	295,484
Number of groups	10,553	10,553	10,553
Wald chi2(16)	9,343	9,191	9,311
Log likelihood	-297,008	-297,089	-297,023

\*: p < 10%    \*\*:p < 5%    \*\*\*: p < 1%

in projects with Production and Mature status. A Wald test shows there is a significant difference in spillovers between projects in Beta and Production stages ( $\text{Chi}^2 = 38.72$ ). It thus appears that spillovers are not a critical resource to start new projects, but rather to improve and extend more mature projects. This suggests that spillovers could relate to bug fixing or generic functionalities rather than the core of new software. Since they mainly benefit the developers of mature projects, this result also provides an interesting explanation for the non-decreasing returns to scale we observe in SourceForge projects.

We fail to find a significant difference between spillovers from projects with the same topic and programming language and spillovers from projects with different programming languages in model 4. Separating the proxies further, however, shows that spillovers for projects with the same programming language are significantly stronger than for projects with different languages (model 6). A Wald test reveals that there is a significant difference between spillovers of same versus different programming languages in production-stage projects ( $\text{Chi}^2 = 4.45$ ) and mature ones ( $\text{Chi}^2 = 7.6$ ). Moreover, we find significant spillover effects from projects with the same programming language, but different topics in models 5 and 6. In model 5, the difference between the spillover effects is significant for same topic-different programming language spillovers and different topic-same programming language.

Finally, we find significant spillover effects from projects with the same programming language, but different topics in models 5 and 6. In model 5, the difference between the spillover effects is significant for same topic-different programming language spillovers and different topic-same programming language.

### *Concluding remarks*

The purpose of this paper was to study empirically the production of open source software. Using a panel of 10,553 projects registered on SourceForge over a period of 28 months (February 2005 until May 2007), we have estimated a production function relating the number of file releases with the number of corporate, private and academic contributors. We have considered two possible specifications of the production function, namely a Cobb-Douglas function

and a more flexible Translog specification. This approach made it possible to highlight various interesting results, concerning the productivity of corporate and other developers, the effects of their interactions within open source projects, returns to scale driven by labor division and the existence of spillovers between projects.

Our first findings concern the developers' productivity. We find empirical evidence that corporate developers are generally more productive than voluntary ones. However, this result must be balanced with the negative effect of interactions between the two categories of developers. In other terms, although corporate developers are more efficient individually, they seem less efficient in cooperating with other categories of developers within a given project, which suggest possible conflicts or at least coordination failures due to vested interests and/or different ways of approaching their work.

Our estimations suggest that open source projects are hardly subject to decreasing returns to scale. This result should be taken with caution due to a lack of data on the time developers spend on each project. Still, it denotes a striking contrast with decreasing returns that are observed in more traditional software production (Brooks, 1978). This suggests that the peculiar organization of production in open source projects – based on a combination of user-driven innovation, voluntary contribution and modular design – enables an efficient division of labor between programmers. Such organizational efficiency may explain why large open source projects such as Linux or Apache are particularly fierce competitors for proprietary software.

Finally, we find evidence for positive spillovers between projects of the Source Forge repository. We test different possible channels, and show that spillovers mainly flow between projects with the same topic and, to a lesser extent, between projects with the same programming language. Projects involving corporate and academic developers are more likely to benefit from spillovers. Project development stage also matters: spillovers only seem to benefit mature projects, thereby increasing further the total productivity of large projects. Surprisingly, the license type neither favors nor hinders the flow of spillovers between projects.

## References

- Amabile, T. (1983). The social psychology of creativity: A componential conceptualization. *Journal of Personality and Social Psychology*, 45(2), 357–376.
- Belenzon, S. & Schankerman, M. A. (2008). Motivation and sorting in open source software innovation. *SSRN eLibrary*.
- Benkler, Y. (2002). Coase’s penguin, or, linux and the nature of the firm. *Yale Law Journal*, 112(3), 367–445.
- Bonaccorsi, A. & Rossi, C. (2003a). Contributing to the common pool resources in open source software. a comparison between individuals and firms.
- Bonaccorsi, A. & Rossi, C. (2003b). Licensing schemes in the production and distribution of open source software: An empirical investigation. *SSRN eLibrary*.
- Bonaccorsi, A. & Rossi, C. (2003c). Why open source software can succeed. *Research Policy*, 32(7), 1243–1258.
- Brooks, F. (1978). *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.
- Chesbrough, H. (2003). *Open Innovation: The New Imperative for Creating and Profiting from Technology*. Harvard Business School Press.
- Christensen, L. & Greene, W. (1976). Economies of scales in us electronic power generation. *Journal of Political Economy*, 84, 655–676.
- Dahlander, L. & Magnusson, M. (2005). Relationships between open source software companies and communities: Observations from nordic firms. *Research Policy*, 34(4), 481–493.
- Dahlander, L. & Wallin, M. (2006). A man on the inside: Unlocking communities as complementary assets. *Research Policy*, 35, 1243–1259.

- Eilhard, J. (2008). Open source incorporated. *Cerna Working Paper*.
- Fershtman, C. & Gandal, N. (2008). Microstructure of Collaboration: The Network of Open Source Software. *SSRN eLibrary*.
- Fosturi, A., Giarratana, M., & Luzzi, A. (2008). The penguin has entered the building: The commercialization of open source software products. *Organization Science*, 19(2), 292–328.
- Ghosh, R., Glott, R., Krieger, B., & Robles, B. (2002). *Free/Libre and Open Source Software: Survey and Study (FLOSS), Final Report, Part 4: Survey of developers*. Technical report. available at: <http://www.infonomics.nl/FLOSS/report>.
- Griliches, Z. & Mairesse, J. (1998). Production functions: The search for identification. (pp. 169–203).
- Hausman, J., Hall, B., & Griliches, Z. (1984). Econometric models for count data with an application to the patents-R&D relationship. *Econometrica*, 52(4), 909–938.
- Henkel, J. (2006). Selective revealing in open innovation processes: The case of embedded linux. *Research Policy*, 35(7), 953–969.
- Henkel, J. (2008). Champions of revealing - the role of open source developers in commercial firms. *SSRN eLibrary*.
- Howison, J. & Crowston, K. (2004). The perils and pitfalls of mining sourceforge. *Proceedings of the International Workshop on Mining Software Repositories (MSR 2004)*, (pp. 7–11).
- Lakhani, K. & Wolf, R. (2005). Why hackers do what they do: Understanding motivation and effort in free/open source software projects. In J. Feller, B. Fitzgerald, S. A. Hissam, & K. R. Lakhani (Eds.), *Perspectives on Free and Open Source Software* chapter 1, (pp. 3–22). MIT Press, Cambridge.
- Lerner, J. & Tirole, J. (2002). Some simple economics of open source. *Journal of Industrial Economics*, 50(2), 197–234.

- Lerner, J. & Tirole, J. (2005). The economics of technology sharing: Open source and beyond. *Journal of Economic Perspectives*, 19(2), 99–120. Retrieved from <http://ideas.repec.org/a/aea/jecper/v19y2005i2p99-120.html>.
- Marschak, J. & Andrews, W. (1944). Random simultaneous equations and the theory of production. *Econometrica*, 12(3), 143–172.
- Olley, G. & Pakes, A. (1996). The dynamics of productivity in the telecommunications equipment industry. *Econometrica*, 64(6), 1263–1297.
- Raymond, E. (2001). *The cathedral and the bazaar. Musings on Linux and Open Source by an accidental revolutionary*. O'Reilly.
- Scotchmer, S. & Maurer, S. (2006). Open source software: The new intellectual property paradigm. In T. Hendershott (Ed.), *Handbook of Economics and Information Systems* (pp. 285–319). Elsevier, Amsterdam.
- Shah, S. (2006). Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science*, 52(7), 1000–1014.
- Singh, P. V., Tan, Y., & Mookerjee, V. (2008). Network effects: The influence of structural social capital on open source project success. *SSRN eLibrary*.
- UNU-MERIT (2006). *Study on the: Economic impact of open source software on innovation and the competitiveness of the Information and Communication Technologies (ICT) sector in the EU*. Technical report, UNU-MERIT, the Netherlands.
- Valloppillil, V. (1998). Halloween documents. *Internal Strategy Microsoft Memorandum*. available at: <http://www.catb.org/esr/halloween/halloween1.html>.
- von Hippel, E. & von Krogh, G. (2003). Open source software and the "private-collective" innovation model: Issues for organization science. *Organization Science*, 14(2), 209–223.

von Krogh, G., Spaeth, S., & Lakhani, K. R. (2003). Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7), 1217–1241.