

Investigating the Productivity of Open Source Software Developers

Kirsten Haaland, Sulayman K. Sowe, Rishab Ghosh and Paul A. David
Collaborative Creativity Group
UNU-MERIT
Keizer Karelplein 19
6211TC Maastricht, The Netherlands

Abstract

The work we present in this workshop is an abstract describing work in progress. Investigating the underlying principles of any domain knowledge is problematic in the sense that many factors coming into play when trying to simplify or implement basic concepts and principles, often are presumed to be understood by all experts researching that domain. Our task at hand is to investigate the productivity of opens source software developers in a sample of FLOSS project in the FLOSSMETRICS database. Data from both CVS logs and a survey we conducted with a cohort of 159 developers was used to produce and statically validate a productivity function, which will attempt to explain or assert that given a certain set of characteristics, developer A has a given productivity in project A'. Having supporting statical evidence to make such an assertion, we hope to extend and build on that model to derive a productivity function which will explain and predict, within the limits of certain statical errors and based on the precision of our data and methodology, FLOSS developers' productivity. As a work in progress, our presentation hopes to leverage the expert opinions in this work to critique, discuss, and share the concept, the characteristics of data sets, proposed methodology, and initial findings as it pertains to investigating the productivity of open source developers.

Keywords

Open Source Software Development, FLOSS, SLOC, CVS, Survey, Productivity.

1. Introduction

Software productivity has been one of the most studied aspects of software engineering (Scacchi 2005). Productivity is generally defined as a ratio of inputs to outputs. The reasons for software productivity attracting so much attention, is the obvious economic importance, where productivity is intrinsically linked with the subject areas cost and effort estimation; in essence it presents the same phenomena in multiple ways. Overall, software productivity finds itself at the intersection between software engineering and economics.

Software production is a complex process, and software productivity covers a wide subject area, including the software product, the software production process and the production setting. This further includes but not limited to programmer/developer productivity, streamlining of the production process such as identifying bottlenecks, studying and accounting for the quality and complexity of code and process. In any productivity study, the challenge is to clearly define, identify and measure the inputs and the outputs. Some aspects of productivity are easier to quantify than others, however it does not follow that it necessarily is the best indicator. This study is specifically investigating the productivity of the *primary* production of software; that is, the time that specifically was spent coding is separated from the time spent on other activities. This is an attempt to counter a trend where input in terms of time could be related to output in so many domains relevant to participating in a community.

The paper is organized as follows: section 2 gives a (brief) overview over the background and related work, followed by section 3 describing the methodology for this study, section 4 describes the data set, section 5 presents some findings, and finally section 6 contains the discussion and conclusion.

2. Background and Related Work

Software cost estimation models

Following the classification of software cost estimation techniques as outlined by Shepperd (2007), three main categories of estimation techniques are identified, namely:

1. Algorithmic or parametric models
2. Machine learning methods (i.e. induced prediction)
3. Expert judgment

In another paper (Jørgensen and Shepperd 2007), a even more detailed breakdown of estimation approaches are identified, namely: regression, analogy, expert judgment, work break-down, function point, classification and regression trees, simulation, neural network, theory, Bayesian and combination of estimates. They show that while regression still is most popular, machine learning methods are on the rise. Also, in general the diversity of estimation techniques is increasing over time.

Turning to the algorithmic and parametric models; COCOMO as developed by Boehm (1981) is probably the most well known method, later an improved version called COCOMO-II was developed that requires the setting of parameters. For an overview of loglinear models including the number of projects (ranging from 15 to 63), the size of the projects measured in size by SLOC and Function Points, as well as the degree of economies or diseconomies of scale, see Banker and Kemerer (1989). Generally software cost estimation models have investigated the productivity of software for a relatively low number of projects, all developed in a firm setting, and actually most cost and effort estimation models are designed for in-house development. This makes the application of these estimation methods problematic for FLOSS projects, as the setting and scope are typically not comparable. Thus when for example applying COCOMO to FLOSS projects, one calculates the substitution costs of what it would have cost to produce the same code in a proprietary setting, such as estimating the substitution cost of Debian using COCOMO in FLOSSImpact (2006); which is not what it actually did cost to produce the code in a distributed collaborative way. The same report also shows the distribution of code output by individuals, firms and universities (p. 50), and affiliation in terms of motivation and productivity could make a difference. Mockus et.al (2003) develops a parametric model for predicting effort changes and how it is distributed over time. Further, within the mainstream of software cost estimation models, Function Points (FP) are becoming more widely used, often replacing LOC or SLOC in effort estimations. For example see Premraj et. al. (2004) for a discussion on productivity over time using FP on a Finnish dataset on proprietary projects. Function Points weights function types in the measure. It in principle addresses some of the issues associated with using SLOC, such as programming language, functionality, complexity and quality. However, obviously using Function Point analysis for FLOSS software is not feasible, and SLOC is still a viable alternative for sizing open source. Investigating the relationship between FP and SLOC in the ISBSG database, a database that reports FP and SLOC for about 400 proprietary projects, we have previously found that there is a strong relationship, which increases the confidence in the measure.

In later years, there has been a shift from algorithmic and parametric models to machine learning methods, such as implemented by Bibi et.al. (2008). Using machine learning methods is promising since it does not necessitate the assumption of a functional form or structure of the relationship in the same way as parametric models do. Finally, surveys show that human centric prediction techniques, also known as expert judgment, is among the most wide spread prediction technique, however this is an under-researched area, even though it is on the rise. (Shepperd 2007, Passing and Shepperd 2003). For a very extensive overview of software development cost estimation studies, see Jørgensen and Shepperd (2007).

Increasing or decreasing returns to scale

Increasing, decreasing or constant returns to scale tend to cause discussion, and the empirical evidence is inconclusive (Shepperd 2007, Kitchenham 2002). However, overall it is commonly assumed that software exhibits diseconomies of scale for proprietary project, on the other hand Dolado (2000) has analyzed existing production functions by

use of genetic programming algorithm, and finds no support for complex production functions.

Other output

Output of participation in a FLOSS community does not necessarily involve the production of code. Community members can for example identify bugs or document software, among many other things. See FLOSSImpact (2006), Lakhani and Wolf (2005) and David and Shaprio (2008) for overviews of motivations in FLOSS projects.

Estimating Proprietary project versus open source

As open source software is growing in size and importance, it has not been easy for researchers to replicate or conduct comparable productivity studies in the open source domain. The reason for this is related to the difficulty of obtaining data, specifically measures of input entering into the production process. This is easier in a traditional company setting, because the company has (some) control over the employees, where the software development happens in-house. In a traditional software production setting, i.e. in a firm, the management or researchers can get programmers to self-report, or a team manager can measure it, or an outside analyst or observer can be hired, or automated performance monitors can be installed to monitor employees, just to mention some alternatives (Scacchi 1995). However, the primary output, being the software product and its corresponding code, is available in both cases.

In other words, for studies focusing on a proprietary setting it has been more feasible to measure both input and output, while for open source software it is generally only possible to observe the output of the production process, the code itself. Further, and ironically, due to the nature of proprietary software licensing, the software output is not generally available to researchers, and this also makes it difficult to compare studies done on proprietary software productivity vs. open source software productivity. Further, the cost estimation studies and databases for proprietary projects are based on completed software development projects, not entirely appropriate for FLOSS as it often includes incomplete projects, developed outside firms.

The output of programming activities is mostly readily available through public repositories and software version control data (“CVS”). However, all these available data sources are output measures, and to calculate the productivity we need to relate the output to the input. Further, one needs to take account of the amount of time and effort the programmer spends on various other activities, not all directly related to coding, such as communicating on mailing lists and writing documentation.

3. Research Methodology

Our research methodology aims to overcome the problems inherent in investigating productivity in the FLOSS context;

Productivity has, as input, a time dimension. One needs to have the actual time a developer spend on a given task. The volunteering nature in FLOSS development makes it difficult for developers to clock, or remember, the exact time they spend on a given

task. The problem becomes even more acute if a researcher wants to obtain times which goes a bit further from the current activity. For example, a developer may find it difficult to remember how many hours he/she spend coding 2-3 weeks ago (time when question is being asked). It is not possible to use CVS check-ins, because artifacts like this are not accurate for input estimation.

As input to investigating productivity is the artefact worked on. This can range from anything on LOC produce, documentation, testing, participating in mailing lists and forums, packaging software, etc. Cyberacrchelogy enables FLOSS researchers to use the wealth of information described as software trail to study who produces which artefact, how much of it. The critical factor is the 'when' the artifact was produced. In software repositories, we can record and datamine times when a certain software artefact was produce but this only tells use when the artifact first appeared or checked-in in the repository and NOT how much time the individual spent producing that artefact. Although, the availability of FLOSS data, lots of it in fact, has leveraged some of the empirical challenges researchers faced, it is now apparent that a lot of effort is needed in terms of data cleaning and aggregation. Our research is no exception to this challenge

In an attempt to overcome these challenges in our productivity study, we adopted what we call "data convergence research methodology" or DCRM. This simply means combining two or more data sources which appear to offer a means to investigate a common research theme. We are currently strongly arguing for DCRM to investigate many of software's complex issues. For example, to really understand the dynamics and evolution of FLOSS communities or 'bug communities' it is important that researchers consider and attempt to integrate data from more than one repository (mailing lists, CVS, BTS) alone.

Our DCRM makes use of two data sources; projects CVS repositories and a survey with developers.

CVS data:

We downloaded CVS/SVN data dumps of 323 projects from the FLOSSMETRICS database. Historically, in 2006 was just when the FLOSSMETRICS started harvesting data from SourceForge.net. The data was processed to obtained primary and derived (computed from the primary) metrics which we assume to be vitals measures in computing our productivity equation. It contains the lines added and deleted per person and per project for the last year (at the point the survey was performed).

Survey data:

For the FLOSS productivity survey we contacted, via email, 8370 committers to the hundred largest FLOSS projects in our data set soliciting their unflinching support to participate on an online survey. We received 460 (approximately 6%) 'willing to participate' emails from these developers, saying that they would be willing to take part in our survey. Subsequently, email was sent to each developer with URL pointer to the

online questionnaire. The questionnaire (available on request) contained 10 questions, covering topics such as the activities of the respondents in the FLOSS community, the time they belong to the project community, the time they spend in producing FLOSS code for the project, trends in time spending in the project, monetary rewards, and demographics. The questions were worded in a way that it was absolutely clear to the respondent to which particular FLOSS project his answers had to relate. To participate the respondents had to provide their CVS id, ensuring they are real developers, as well as being able to match the reported time input reported in the survey with the CVS code output. Out of the 460 developers who were willing to participate on our survey, 159 completed the survey, corresponding to a response rate of about 45%.

4. Initial Data, results and analysis.

CVS data

Table 1 shows some descriptive statistics of our primary metrics that we obtained from CVS logs that we are investigating. These preliminary observations were necessary to enable us map the appropriate analysis technique to a given metric in the data set.

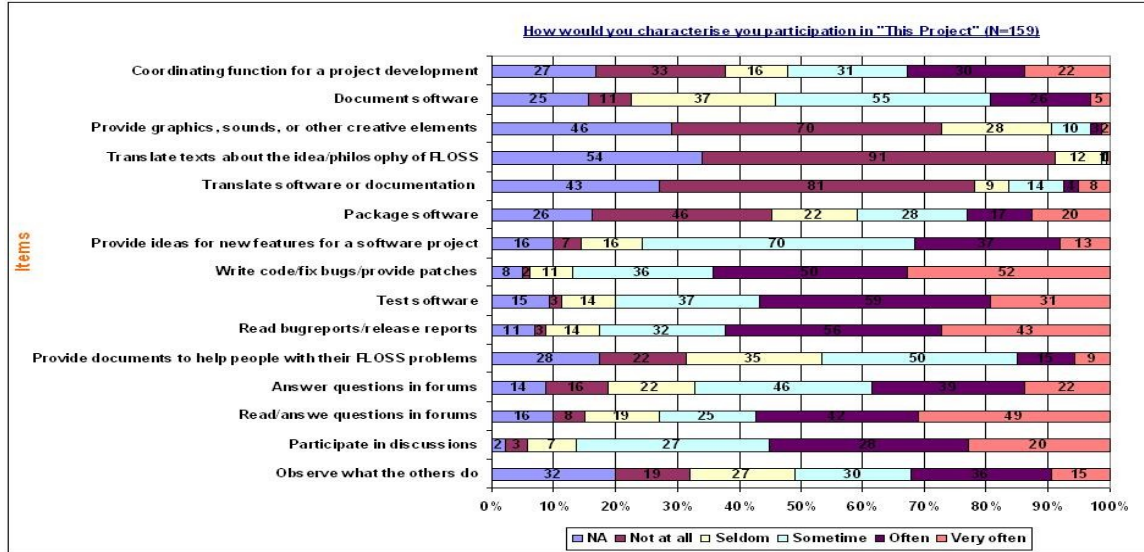
Table 1: CVS data individuals

	Obs	Mean	Std. Dev.	Min	Max
plus	8370	52900.59	284938.5	0	9557726
minus	8370	34212.11	206360.9	0	7637788
net	8370	18688.49	112599.7	-340758	4213076
files	8370	375.84	1285.23	1	51691
commits	8370	1187.19	4360.06	1	143006
weekly_gross_output	2384	896.75	21649.99	0.02	1053295
Net to gross ratio	7344	0.26	3.99	-316	1
plusbycommit	8370	46.4	174.45	0	8663.53
netbycommit	8370	17.53	134.14	-2265.41	8655.76

Survey data:

For example, the stacked bar charts in the figure below compares the percentage response to a battery of questions we asked the developers in Question 2 of the survey.

Table 2: “other” developer activities



5. Preliminary findings

The main aim of the productivity survey was to determine how much time FLOSS developers *actually* spend, and how much they produce in this time. In other words the real cost and the real productivity. This has proved to be very challenging as we are finding low correlations so far, shown in the table below.

Table 2: Correlations

			Rank of weekly_gross_output	Rank of q3a
Kendall's tau_b	Rank of weekly gross output	Correlation Coefficient	1.000	.288(**)
		Sig. (2-tailed)	.	.000
		N	87	87
Spearman's rho	Rank of coding	Correlation Coefficient	.288(**)	1.000
		Sig. (2-tailed)	.000	.
		N	87	87
Kendall's tau_b	Rank of weekly gross output	Correlation Coefficient	1.000	.401(**)
		Sig. (2-tailed)	.	.000
		N	87	87
Spearman's rho	Rank of coding	Correlation Coefficient	.401(**)	1.000
		Sig. (2-tailed)	.000	.
		N	87	87

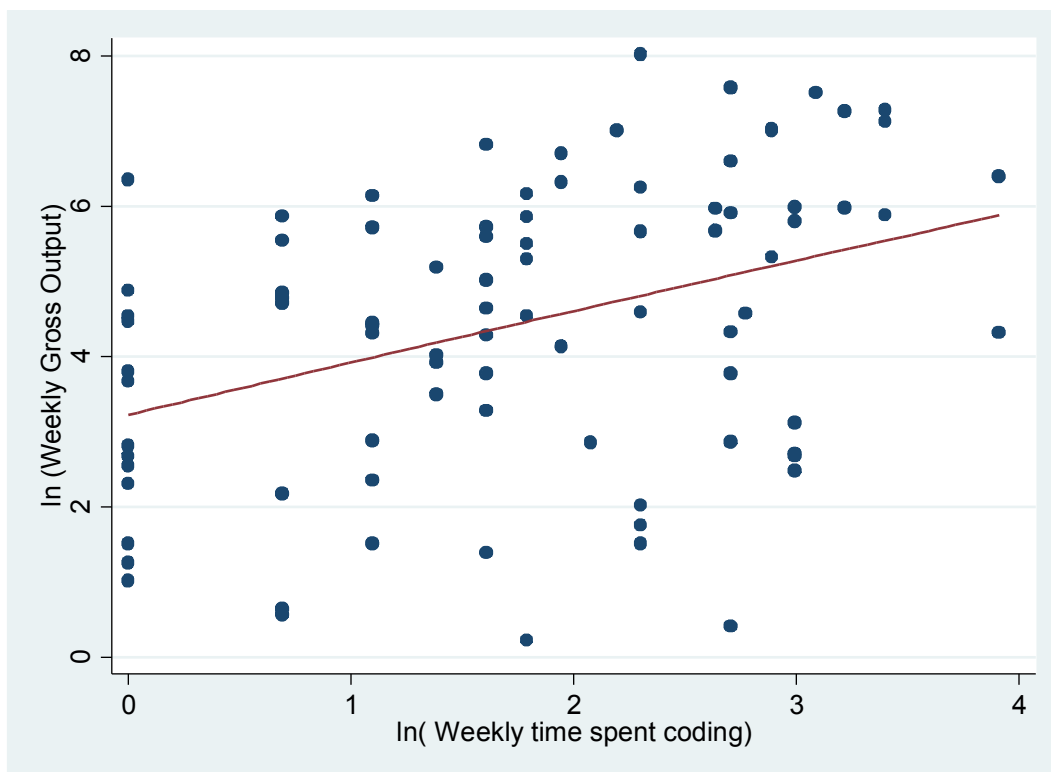
** Correlation is significant at the 0.01 level (2-tailed).

Figure 1 shows results of the curve estimation regression. The R^2 value indicates a rather weak relationship between developers weekly gross output and the actual time spent coding. A linear regression model explains only 14,6% ($R^2 = 0.146$) of the variability. However, a quadratic fit method was not able to improve the relation significantly ($R^2 = 0.147$). The equation for the linear regression can be expressed thus:

$$y = 3.23 + 0.68 x$$

where y = Developer Weekly Gross and x = Output developer coding time.

Figure 1: Linear regression



6. Conclusion

The future work involves the development of a spot productivity estimator function based on the survey data combined with the CVS data. Our current analysis is a form of an exploratory study to see which kinds of variables can be combined to give the best model fit. We have come to realize that any form of productivity function that we may come up with must take into consideration many factors come into play when we consider developer productivity. For example, we have seen a significantly improvement R^2 value when we introduced the element of paid contribution to FLOSS development. This is the

case for those developers who were asked, in the survey, whether they are directly or indirectly paid for their contribution. Other elements we are trying to introduce into our equation, are time spent on other activities, such as software testing, bug fixing, forum activity and coordination etc. All these activities received high percentage scores from the developers in our survey. Further, we need to investigate measures such as a developers ranking in terms of long term contribution and activity. Another promising source is project characteristics, such as overall size and age of the project.

Acknowledgments

This work is supported by the Free/Libre/Open Source Software Metrics and Benchmarking Study (FLOSSMetrics), Project No: 033982, (<http://flossmetrics.org>).

References

Banker, R. D., Chang, H., and C. F. Kemerer, 1994, "Evidence on economies of scale in software development," *Info. & Softw. Technol.*, 36, pp 275-282.

Banker, R. D. and Kemerer, C. F. 1989, "Scale Economies in New Software Development", *IEEE Transactions on Software Engineering*, vol. 15 (10), p.1199-1205.

Bibi, S., Stamelos, I., and Angelis, L. 2008, "Combining Probabilistic Models for Explanatory Productivity Estimation", *Information and Software Technology*, Elsevier, 50(7-8), pp. 656-669.

Boehm, B. 1981, *Software Engineering Economics*, Prentice Hall PTR, Upper Saddle River, NJ.

David and Shaprio, 2008, "Community-Based Production of Open Source Software: What do we know about the developers who participate?" Available at SSRN: <http://ssrn.com/abstract=1286273>

Dolado, J. 2001, "On the problem of the software cost function". *Information & Software Technology*, 43(1) pp 61-72.

Ghosh et. al. 2006, FLOSSIMPACT: The impact of Free/Libre/Open Source Software on innovation and competitiveness of the European Union, available online at: <http://www.flossimpact.eu/>

Kitchenham, B. 2002 "The question of scale economies in software - why cannot researchers agree?" *Info. & Softw. Technol.*, 44, pp 13-24.

Lakhani, Karim R. and Wolf, Robert G., Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects (September 2003). MIT Sloan Working Paper No. 4425-03.

Mockus, A., Weiss, D. M., and Zhang, P. 2003. Understanding and predicting effort in software projects. In Proceedings of the 25th international Conference on Software Engineering (Portland, Oregon, May 03 - 10, 2003). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 274-284.

Jørgensen, M., and Shepperd, M., 2007 A Systematic Review of Software Development Cost Estimation Studies, IEEE Transactions on Software Engineering, v.33 n.1, p.33-53.

Passing, U. and Shepperd M., 2003: An experiment on software project size and effort estimation. ISESE 2003: 120-131

Premraj, R., Shepperd, M., Kitchenham, B. and Fordselius, P. 2005. An empirical analysis of software productivity over time. In METRICS '05: Procs. of the 11th International Software Metrics Symposium, page 37, Como, Italy, IEEE.

Scacchi, W. "Understanding software productivity," in Advances in Software Engineering and Knowledge Engineering, vol. 4, 1995, pp. 37-70

Scacchi, W. 1995. Understanding and Improving Software Productivity, Institute of Software Research, presentation online available: www.ics.uci.edu/~wscacchi

Shepperd, M. 2007. Software project economics: a roadmap. In 2007 Future of Software Engineering (May 23 - 25, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 304-315.