What Does It Take to Develop a Million Lines of Open Source Code?

Juan Fernandez-Ramil[1,3], Daniel Izquierdo-Cortazar[2], and Tom Mens[3]
[1] Université de Mons
Mons, Belgium
{j.f.ramil, tom.mens}@umons.ac.be
[2] Universidad Rey Juan Carlos
Madrid, Spain
dizquierdo@gsyc.urjc.es
[3] The Open University
Milton Keynes, U.K.
j.f.ramil@open.ac.uk

Abstract. This article presents a preliminary and exploratory study of the relationship between size, on the one hand, and effort, duration and team size, on the other, for 11 Free/Libre/Open Source Software (FLOSS) projects with current size ranging between between 0.6 and 5.3 million lines of code (MLOC). Effort was operationalised based on the number of active committers per month. The extracted data did not fit well an early version of the closed-source cost estimation model COCOMO for proprietary software, overall suggesting that, at least to some extent, FLOSS communities are more productive than closed-source teams. This also motivated the need for FLOSS-specific effort models. As a first approximation, we evaluated 16 linear regression models involving different pairs of attributes. One of our experiments was to calculate the *net size*, that is, to remove any suspiciously large outliers or jumps in the growth trends. The best model we found involved effort against net size, accounting for 79 percent of the variance. This model was based on data excluding a possible outlier (Eclipse), the largest project in our sample. This suggests that different effort models may be needed for certain categories of FLOSS projects. Incidentally, for each of the 11 individual FLOSS projects we were able to model the net size trends with very high accuracy ($R^2 \geq 0.98$). Of the 11 projects, 3 have grown superlinearly, 5 linearly and 3 sublinearly, suggesting that in the majority of the cases accumulated complexity is either well controlled or don't constitute a growth constraining factor.

Keywords: baselines, complexity, COCOMO, cost, economics, effort estimation, effort operationalisation, empirical studies, large software, free software, metrics, open source, productivity, software evolution.

1 Introduction

Software development productivity and cost estimation have been research topics for more than three decades (e.g., [21] [3] [18]). The vast majority of these studies in-

volved data from closed-source projects. Little seems to be known about effort models for large, long-lived FLOSS projects. The present paper[1] presents our initial exploratory results in the study of the relationship between size, effort, duration and number of contributors in eleven large, long-lived FLOSS projects.

FLOSS communities do not use, in general, effort estimation models or other measurement-based models [1]. As a research topic, the exploration and quantification of effort and productivity may help in comparing FLOSS projects and proprietary systems. This research may provide a basis for baselines, in order to evaluate the possible impact of changes of processes, methods and tools used by FLOSS communities. It may also help, in some way, in understanding and better planning the future evolution of FLOSS communities and their projects[2].

Before presenting our approach and results, an important limitation of this type of research must be mentioned. It has been pointed out (e.g., [11] [13]), that it is difficult to measure effort accurately in FLOSS. This is so for several reasons: for example, we do not know the degree of involvement of each contributor (Was it full-time? Was it some level of part-time?) and contributors may have widely different roles. Despite our best efforts to measure effort in a meaningful way, we are aware of the limitations of our approach and this is why the results reported here must be seen as exploratory and preliminary. Additional research is needed to achieve more reliable measures of effort involved in FLOSS projects. We hope, however, that our contribution can help establish FLOSS effort modelling as a line of research and, eventually, lead to scientific insights and practical tools for the FLOSS communities and other interested stakeholders.

2 Selected projects and measurements

Our research question was how much effort would be required to develop (via evolution) a FLOSS project of one MLOC (i.e., 1000 KLOC). We chose eleven projects, all of which exceed 1 MLOC (one 0.6 MLOC project in our sample has achieved this size at some point before decreasing) and have code repositories compatible with our data extraction tools. Table 1 shows the names of the eleven FLOSS projects we considered, the programming language(s) primarily used for each system, a short description and the life span, that is, the time in years between the first publicly recorded commit to the code repository (CVS or Subversion) and the data extraction date. This was October 2008, with exception of one system (Eclipse) for which it was April 2008.

The measurements extracted for this study are listed in Table 2. We used the SLOCCount tool[3] to measure lines of source code. It automatically identifies code developed in a variety of programming languages and, for each of them, counts the lines of code present in a directory. We used CVSAnalY[4], which stores information extracted from the version control log (CVS or Subversion) in a MySQL database. Specifically, we

[1] This is a revised and extended version of [7].

[2] See [12] for a fuller justification for research into FLOSS effort models

[3] `www.dwheeler.com/sloccount/`

[4] `svn.forge.morfeo-project.org/svn/libresoft-tools/cvsanaly`

Table 1. FLOSS systems studied and some of their characteristics

name	primary language	description	life span in years
Blender	C/C++	cross-platform tool suite for 3D animation	6
Eclipse	Java	IDE and application framework	6.9
FPC	Pascal	Pascal compiler	3.4
GCC	C/Java/Ada	GNU Compiler Collection	19.9
GCL	C/Lisp/ASM	GNU Common Lisp	8.8
GDB	C/C++	GNU Debugger	9.5
GIMP	C	GNU Image Manipulation Program	10.9
GNUBinUtils	C/C++	collection of binary tools	9.4
NCBITools	C/C++	libraries for biology applications	15.4
WireShark	C	network traffic analyser	10
XEmacs	Lisp/C	text editor and application development system	12.2

Table 2. Measured attributes for each system

abbreviation	description	extracted using
KLOC	physical lines of source code (in thousands)	SLOCCount
FILES	total number of files in code repository	CVSAnalY
EFFORT	'contributor-years' (based on active committers-month)	CVSAnalY
DUR	time length of 'active' evolution in years	CVSAnalY
DEV	number of distinct contributors	CVSAnalY

indirectly used the data from CVSAnalY by downloading databases from the FLOSSMetrics project[5].

In order to *operationalise* the effort variable, we proceeded as follows: first, we ran a query to the CVSAnalY databases for each system to determine the number of different person ids contributing to its repository over a given month. *This value was assumed to be the number of person-months in a given month.* The total number of person-months, added over the lifetime of the code repository, was divided by 12 in order to obtain the equivalent to person-years, which is the value assumed to be the total effort spent into a system since code commits recording started ($EFFORT$). As said in section 1, measuring effort in FLOSS projects is notably difficult. Our effort measurement approach is likely to be, in some cases, an overestimation, since not all the contributors are likely to work full-time on a project (with exception of some company-supported projects). Interestingly, in some projects one or several *gate keepers* submit code developed by others. If present, this will turn our $EFFORT$ values into an underestimation of the real effort. This is clearly a topic for further research. We present our operationalisation of effort as an starting point which can be improved in the future by using additional information about, for example, the social networks and work patterns of contributors in a given project, and eventually by advanced tools which eventually will register automatically the effort.

[5] `data.flossmetrics.org`

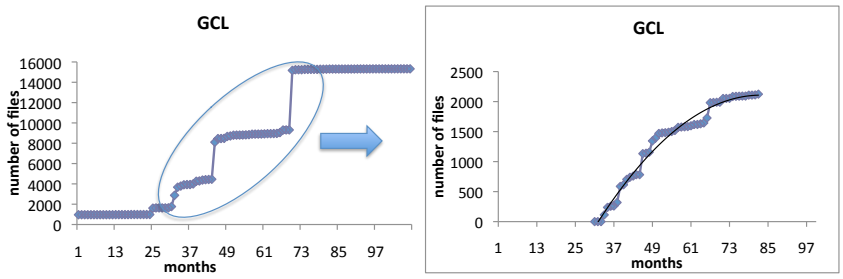

Fig. 1. Trend in FILES (left) and netFILES (right) per month for GCL. The right-hand-side plot corresponds to the central part of the left-hand-side plot, after major 'jumps' have been removed.

In 9 out of 11 projects studied – that is, excluding Eclipse and Wireshark – when looking at the plot of the total number of files ($FILES$) in the repository per month, we observed a few, suspiciously large outliers or *jumps*. This is illustrated for one of the systems (GCL) on the left part of Figure 1. It is unlikely that the team size or the productivity of the team of contributors has increased so suddenly on a particular month. It could be that code added in a jump was created and evolved in a private space (or as a branch separated from the main version or *trunk*) for which we have no active committers information. It could also be that each jump corresponds to external events such as when the repository receives chunks of externally generated code. An additional hypothesis would be that many non-code files were added to the repository, since $FILES$ includes all files, code and non-code.[6] Whatever the actual explanation, the implication for our research is that measuring the total size, including jumps, may lead to too optimistic estimates (the productivity may appear to be higher than it really is). We filtered out the jumps by removing their corresponding size increments, that is, calculating what we call the *net size*. We also subtracted the initial size value, since we do not have any records of effort for this initial commit. Figure 1, on the right, shows the result for GCL. On the left one can observe the trend for $FILES$ with several large jumps. The filtered trend, that we call $netFILES$, is shown on the right.

We measured project duration DUR as the time between the earliest and most recent commit. We excluded periods with no commits or with a number of commits much lower than during other clearly active periods. We did this in order not to penalise projects with periods of inactivity, when modelling the relationship between size and duration. This filtering operation determined the value of DUR that we assumed for the GCL project (4.3 years instead of 8.8 years, as can be seen on the right of Figure 1). We did a similar filtering for GCC, with a resulting value for DUR of 11.2 instead or 19.9 years.

For calculating the number of contributors DEV we obtained the number of people having made at least one commit since the start of the repository. We computed two

[6] SLOCCount automatically discards non-code files but we applied this tool purely to measure the size of the latest repository for each system. It would have been convenient to apply the tool to obtain monthly measurements of size but this was beyond our capability.

Table 3. Best fit models for monthly growth trends in *netFILES*

system	best growth trend model	R^2
Blender	linear	0.99
Eclipse	linear	0.99
FPC	linear	0.98
GCC	superlinear	0.99
GCL	sublinear	0.98
GDB	linear	0.99
GIMP	sublinear	0.99
GNUBinUtils	linear	0.99
NCBITools	superlinear	0.98
WireShark	superlinear	0.99
XEmacs	sublinear	0.98

variants, one measuring all the contributors, called $DEV[100]$, and one measuring the number of developers which provided at least 80% of the effort in contributor-months, called $DEV[80]$. We did this in order to try to distinguish the core team, generating and evolving most of the code, from other less active contributors which may have contributed with only a few items (e.g., defect fixes) over the lifetime of the project.

3 Results

3.1 Growth models

We modeled the *netFILES*, the filtered growth trends, over months. We found a very good fit to linear, quadratic or exponential models, with R^2 values[7] ranging from 0.98 to 0.99, as can be seen in Table 3. It is worth noting that for GCC and GCL, the growth model is fitted only over the period of active evolution). Five of the systems have trends that follow linear models, three follow sublinear (quadratic) models and three superlinear models (2 quadratic and 1 exponential). The high goodness of fit suggests that work in these FLOSS communities happens at a regular, very predictable rate.

The net growth trends can be used as a simple estimation model of the amount of new functionality implemented per month, assuming that there are no radical changes such as, for example, the departure of key members in the core team, or an important architectural restructuring. Careful projection of these growth trend models into the future could be helpful for a given project and can be the basis for *project specific* effort models. In the next sections we examine possible *generic* models, that are based on the data from the 11 FLOSS projects studied and may have potential to be generalised, that is, used across systems.

[7] In this paper we used R^2, the *coefficient of determination*, as our exploratory measure of goodness of fit, where a value of 1 indicates a perfect fit and a value of 0 indicates no fit. Further work should evaluate other measures such as the adjusted-R^2, the mean (or median) magnitude of relative error (MMRE) or the Akaike Information Criterion (AIC).

3.2 Comparison of FLOSS data with the COCOMO model

The COnstructive COst MOdel (COCOMO) is a classic estimation model developed by Barry Boehm and colleagues, based on data from closed source systems. Some of its assumptions are described in [12]. As a starting point, we used here its earliest and possibly simplest version [3], called COCOMO 81, published in 1981. Given that the COCOMO model has itself several variants, we used the default COCOMO predictions generated by default by the SLOCCount tool. In this comparison, we are assuming that COCOMO will be representative *in the range of size values* similar to those of the studied FLOSS systems[8]. Figures 2 to 5 show the studied FLOSS systems as squares, and the COCOMO model as a solid line. In the four figures, the x-axis represents size in *KLOC* and the y-axis represents $EFFORT, DUR, DEV[100]$ and $DEV[80]$, respectively. For illustration purposes and as an initial exploration, we superimposed a linear regression model to the FLOSS data, shown as a dashed line. The figures display the mathematical expressions for COCOMO (power models) and for the linear models based on the FLOSS data. Examining figures 2 to 5 one can identify the following observations:

- 10 out of 11 projects display $EFFORT$ that is less than the one predicted by COCOMO, overall suggesting higher productivity (as KLOC per contributor-month) in FLOSS than in closed-source. Only the project with smallest *KLOC* in the sample (GIMP) is close to COCOMO's prediction. There is evidence in [10] that GIMP grew up to 1 MLOC in the past and then shrinked to its current size (646 KLOC). This could explain why GIMP is the project with the smaller productivity in the sample.
- 8 out of 11 projects display values of DUR which are higher than COCOMO predictions. This may be explained by the fact that the FLOSS projects are evolving systems over multiple releases while COCOMO represents software built from scratch to achieve its first release. The FLOSS projects do not have, in general, a fixed schedule and budget and they continue to evolve as long as there is interest and motivation in their communities.
- 6 out of 11 projects show values of $DEV[100]$ which are higher than COCOMO. When looking at $DEV[80]$, however (i.e., our approximation to the number of people in the core team), only 1 out of 11 projects (namely GIMP) exceeds the team size predicted by COCOMO.
- Overall, COCOMO 81 isn't a good model for FLOSS data.
- The goodness of fit of the linear models fitted to the FLOSS data is poor. The best linear model is the one between $EFFORT$ and *KLOC* with R^2 value of 0.34. This means that only 34 percent of variance in the data is accounted for in the linear model.

[8] Unfortunately we could not check this when finalising this paper. There is a risk that in our analysis we used COCOMO 81 beyond its range of validity.

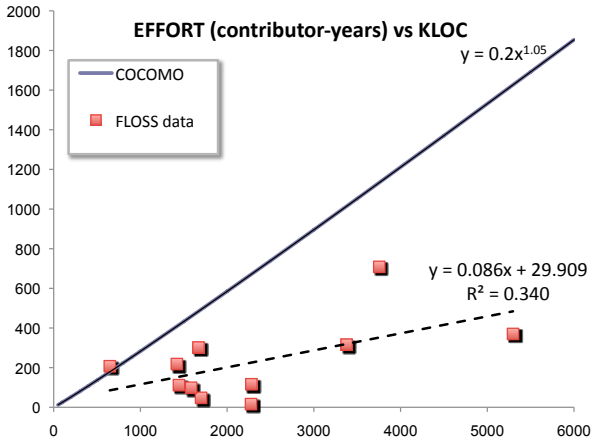

Fig. 2. EFFORT vs KLOC: FLOSS data (N=11) and COCOMO model.

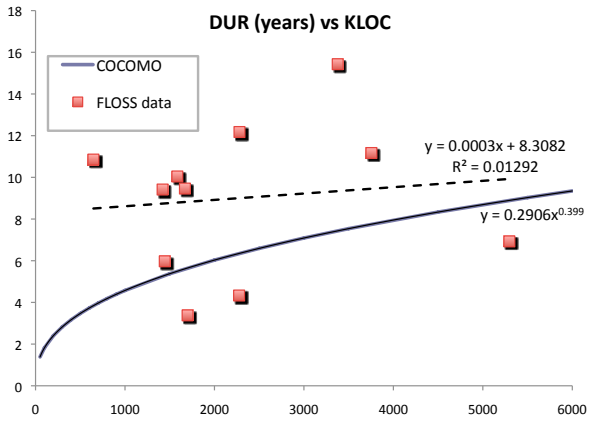

Fig. 3. DUR vs KLOC: FLOSS data (N=11) and COCOMO model.

3.3 FLOSS-based estimation models

In search for possible simple FLOSS-based generic estimation models we explored further the linear correlation between size (measured not only in $KLOC$ but also as $FILES$) and $EFFORT$, DUR and DEV. We also defined $netKLOC = KLOC * \frac{netFILES}{FILES}$ and evaluated whether the net size values ($netFILES$ and $netKLOC$) provided an improvement in the regression results obtained using $KLOC$ and $FILES$.

Tables 4 and 5 show the parameters of linear models of the form $y = (a * size) + b$ and the corresponding R^2 values, for all the systems and excluding Eclipse, respectively. Parameters a and b are not reported when R^2 is less than 0.1 because they are considered as not meaningful. Figure 6 shows a scatter diagram and one of the linear

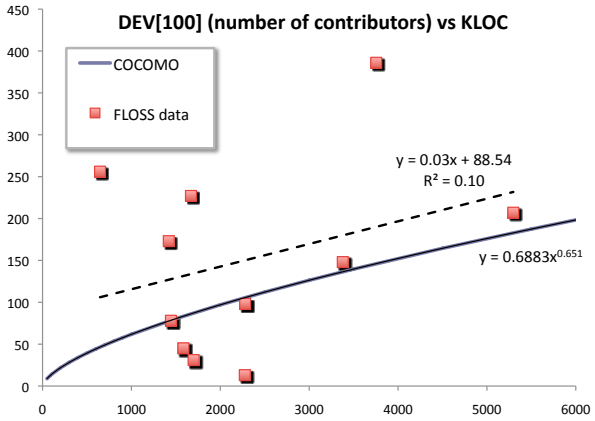

Fig. 4. DEV[100] vs KLOC: FLOSS data (N=11) and COCOMO model.

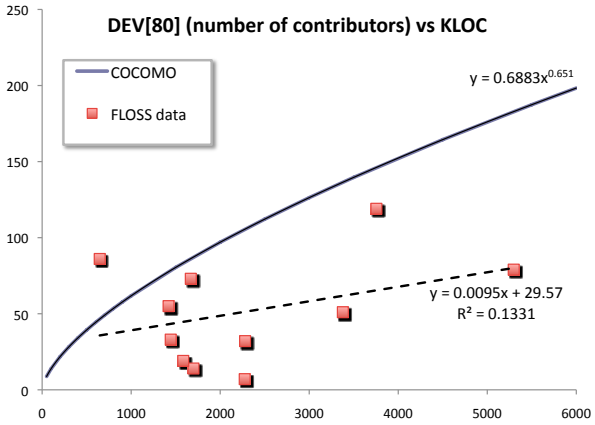

Fig. 5. DEV[80] vs KLOC: FLOSS data (N=11) and COCOMO model.

models superimposed as a line. In this Figure, it is possible to appreciate that Eclipse, which is represented by the square closest to the right side of the plot, is one of the possible outliers. 13 out of 16 models improved, in terms of R^2, when Eclipse was excluded from the dataset. In Tables 4 and 5, best models are indicated in bold. The best linear model, shown in Figure 7 is the one involving $EFFORT$ as a linear function of $netFILES$, excluding Eclipse (R^2 value of 0.797).

These results suggest that it may be helpful to have a separate set of models for FLOSS systems with particular characteristics (e.g., very large systems, such as those greater than 5 MLOC, or systems based on different variants of the FLOSS processes – company-led as opposed to community-led projects, – or particular technology, since Eclipse was the only Java-based system in the sample).

Table 4. Linear regression results - parameters a, b and R^2 values

	EFFORT a, b, R^2	DUR - duration a, b, R^2
including Eclipse		
KLOC	0.086, 29.909, 0.339	-, - , 0.012
FILES	0.0039, 119.58, 0.390	-, -, 0.001
netKLOC	0.076, 110.39, 0.326	- , - , 0.048
netFILES	0.0035, 156.61, 0.313	- , - , 8.8E-05
excluding Eclipse		
KLOC	0.1327, -53.323, 0.387	0.0015, 6.1233, 0.153
FILES	0.0093, 3123, 0.66	- , - , 0.06
netKLOC	0.1699, 14.247, 0.499	0.0032, 5.4474, **0.525**
netFILES	0.0139, 51.455, **0.797**	- , - , 0.094

Table 5. Linear regression results -results for DEV metric

	DEV[100] a, b, R^2	DEV[80] a, b, R^2
including Eclipse		
KLOC	0.027, 88.54, 0.101	0.0095, 29.57, 0.133
FILES	0.0017, 103.97, 0.219	0.0006, 35.14, 0.285
netKLOC	0.0287, 106.56, 0.139	0.0105, 35.38, 0.196
netFILES	0.0016, 119.2, 0.185	0.006, 40.07, 0.258
excluding Eclipse		
KLOC	- , - , 0.088	- , - , 0.07
FILES	0.004, 62.831, 0.391	0.0012, 24.787, 0.367
netKLOC	0.0626, 71.879, 0.196	0.0183, 27.401, 0.185
netFILES	0.0143, -19.009, **0.565**	0.002, 25.998, **0.506**

Contrary to what was expected, the removal of jumps in the monthly growth of the studied systems did not lead to visible improvements in the goodness-of-fit of the linear models. The best models involved $EFFORT$ vs size but still with a low R^2 value of around 0.3. The worst models were obtained for DUR. As shown in Figure 3, the studied FLOSS projects do not seem to have achieved its current size at similar rates.

3.4 Calculating the 'cost' of 1 MLOC in FLOSS

In order to address our research question given in the title of this paper and, more precisely, in Section 2, we present some simple calculations, based on the best models identified in the previous section. Since our best model is based on file counts, we need to convert 1 MLOC into number of files. For the 11 FLOSS studied, the file size varied between 50 and 284 lines of code, with an overall average of 125 lines of code per file.

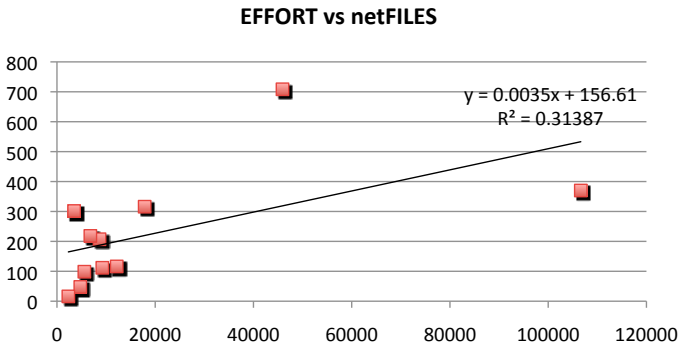

Fig. 6. EFFORT vs netFILES: FLOSS data (N=11) and linear regression model.

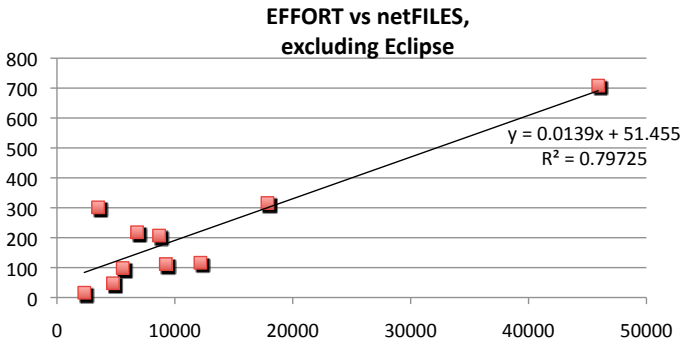

Fig. 7. EFFORT vs netFILES: FLOSS data, excluding Eclipse (N=10) and linear regression model.

Using this average[9], we assume that 1 MLOC will be equivalent to some 8,000 code files. Taking this value and using our best model, the model '$EFFORT$ vs $netFILES$ excluding Eclipse', we obtain an estimate of 162.6 contributor years. Applying the model 'DUR vs $netKLOC$ excluding Eclipse', we calculate an estimate duration of 8.6 years. Dividing 162.6 by 8.6 gives a core team size of 18.8 contributors. Alternatively, if we use the model 'DEV[80] vs $netFILES$' excluding Eclipse, we obtain a team size of 41.9 core developers. Dividing 162.6 by 41.9 gives an estimated duration of 3.8 years. The two values of DEV and DUR can be seen, respectively, as lower and upper bounds of the estimate. These values differ from COCOMO 81 estimates. For a 1 MLOC project, COCOMO 81 predicts the need for 283 contributor-years (higher than FLOSS estimates), with 62 developers (also higher than FLOSS) working during 4.6 years (within range, towards lower end). Our simple models are in line with section 3.2,

[9] This is a simple calculation for illustration. In general, average values of software metrics should either be used with care or not used because data distributions may be highly skewed and in that case averages will not be meaningful [5].

suggesting that FLOSS projects require less total effort and a smaller number of developers (core contributors) than closed-source. Our primitive analysis also suggests that closed-source would perform comparatively better for duration than FLOSS.

4 Related work

In recent years, researchers have increasingly extracted data from FLOSS artefacts, code repositories, defect databases and mailing lists in order to find interesting facts (e.g., [17] [20]). To our knowledge, there have been no previous similar studies of large FLOSS software, attempting to quantify the relationships between size, effort, duration and team size. In a position paper, Amor *et al.* [1] have discussed the problem of effort estimation in a FLOSS project and how classical estimation models may be improved to be applicable to FLOSS projects. Liguo Yu [23] presented an indirect approach to estimate maintenance effort in FLOSS, based on data from NASA SEL closed-source and on Linux. He indirectly measured effort by looking at the number of lines of code (and modules) added, deleted and modified. His work focused on the amount of effort for individual maintenance tasks and, for this reason, is not directly comparable to ours, which focused on total effort at system level. Koch [11] [13] studied the impact of tool usage on the efficiency of 30 FLOSS projects from `SourceForge.net` finding, in most of the cases, an intriguing negative relationship. In a different paper, the same author [12] reported on the testing of two hypotheses involving different types of effort models, including COCOMO, in a sample of more than 8,000 projects from SouceForge.net. Differences in approach and the fact that the vast majority of projects in this study were small (involving less than 10 developers), make this study difficult to compare to ours, which focused on larger projects.

The majority of software estimation research has been conducted for closed-source software and initial development(e.g., [18]). One frequently cited software estimation model, based on closed-source software, is COCOMO [3]. We have compared COCOMO 81 - the earliest and simplest version – to our FLOSS data, and found that it is not appropriate to model the productivity of our subset of large FLOSS projects. Much more research needs to be done (e.g., repeat the comparison using COCOMO II [4] and other more recent models), but our initial finding suggests that FLOSS-specific models are advisable. This is not surprising if one considers the particular characteristics of FLOSS (e.g., [19]).

The proportion of linear, superlinear and sublinear growth trends (cf. Table 3) is generally in agreement with some previous empirical studies of FLOSS growth trends [9]. The predominance of linear and superlinear trends might be questioning the assumed role of accumulated complexity [15] [16] as a growth constraining factor. An alternative hypothesis to be studied is that linearly and superlinearly growing projects are effective at controlling complexity.

As far as we know, our work is the first that explore the impact of removing outliers or jumps in the monthly growth trends. In future work this type of outlier detection needs to be done more systematically than our simple visual detection.

5 Threats to Validity

Our empirical study is subject to many threats to validity. A list of threats that are likely to apply in general to empirical studies of FLOSS is given in [8]. In general, one can identify three set of threats: threats to construct, internal and external validity.

5.1 Construct validity

Our estimation of effort is far from perfect. Accurately measuring effort in open source projects is difficult. We have counted the number of active committers (i.e., developers who have checked in code into the repositories in a given month) and assumed that each contributed with a 'contributor-month'. Many developers are part-time volunteers and others may be paid employees of sponsoring companies working part-time or full-time on a given FLOSS project. It is known that in some cases, code is committed to the repository by a gate keeper, not by the actual developer. This may also bias the effort measures. One way to improve measuring effort would be to conduct a survey of FLOSS contributors to know better their work patterns and use this knowledge to adjust our measurements. Surveys, however, may require considerable research time and resources, including the willingness of FLOSS contributors to participate. Our effort measurement may be more accurate for FLOSS projects like Eclipse, where a portion of contributors are full-time employees of a sponsoring company (in this case, IBM).

Our measures of size (*KLOC* and *FILES*) may include automatic generated code that may have biased the results: the system will be bigger than it should have been if all the code were generated and evolved manually. The measured repositories may include external libraries with or without any modification. Code may be ported in FLOSS communities from one project to another biasing productivity measures. We were not able to quantify how many lines of code have been ported, as opposed to generated from scratch. In addition to this, our *FILES* measure includes all files in the repository. We plan, in the future, to measure the amount of code files only and check whether the modelling results improve. *Cloning*, or code duplication within the same project [2], is also a phenomenon present in open source. Code cloning may increase productivity but may have the opposite effect as simultaneously evolving many clones may slow down progress.

5.2 External and internal validity

The sample of projects that we studied is very small when compared with the total number of open source projects. A popular open source hosting website, `sourceforge.net`, lists currently more than 300,000 projects. However, only a small fraction of the total number of open source projects can be considered successful [6]. Even smaller is the number of projects that have reached 1 MLOC. The sample we studied is not truly random. We selected projects that were feasible to analyse, based on their availability in the FLOSSMetrics[10] project database.

[10] `flossmetrics.org`

The tools we used in this study to extract and analyse the data may contain defects that may have affected the results. We collected, analysed and plotted the data using spreadsheet software which, despite our best efforts, is also error-prone. Unfortunately we did not have enough time for an independent data extraction and analysis but we hope that other researchers will replicate and extend this work in the future. We were not aware of any other possible threats to internal validity apart from the measurement issues already mentioned.

6 Further work

This research could be continued in a number of ways. For example, our *FILES* measurement considers all files in the repository (e.g., code, configuration, data, web pages). It is likely that better results will be achieved by considering code files only. Better results may be also be achievable by using *robust regression* [14]. Modelling techniques different to regression have been tried in classical (i.e., closed-source or proprietary) cost estimation (e.g., [18]) and these could also be applied to FLOSS data. We also would like to exclude any automatically generated files since they will bias the results (e.g., productivity may appear higher than it is). Another further topic is the experimentation with different approaches to extract the outliers in growth trends. In Figure 1 (right) one can identify 'jumps' that were not apparent when looking at the unfiltered data (left). One question is how to define formally what is a growth outlier, identify its nature and how frequent they happen in FLOSS evolution. It wouldn't be surprising if the presence of jumps is typical for FLOSS processes, where external code is 'borrowed' and re-used.

By measuring the amount and type of *refactoring* work (e.g., [22]), code transformations that preserve functionality and decrease complexity, in FLOSS projects and combining this data with effort measures, we may be able to understand better the impact of accumulated complexity and why the size of some projects has been growing linearly or superlinearly.

In order to improve this research (e.g., increase the validity of the findings), one should analyse an additional number of FLOSS projects. The linear models we used are simple and do not reflect any theory about FLOSS. Theories based on a deeper understanding of FLOSS (e.g., [19]) and new metrics (e.g., [10]) may lead to more accurate and helpful effort models.

7 Conclusions

Getting people to develop (via evolution) software systems of one million lines of code (1 MLOC) or more, at no cost, as a hobby or leisure activity, is an impressive achievement of a number of FLOSS communities. Rather than studying motivational factors, in this paper, we took a strictly quantitative point of view by gathering data on elevent large FLOSS systems and studying relationship between size, on the one hand,

and effort, duration and team size. Productivity, for large FLOSS systems like these is a topic which does not seem to have been empirically studied.

The comparison between data from 11 FLOSS projects and an early version of COCOMO [3] suggested that FLOSS might be more effective than closed-source projects. It also suggested that FLOSS-based effort estimation models are needed. Our best model to date, using simple linear regression, leaves still much room for improvement ($R^2 = 0.79$). Better models may be obtained by exploring theory-based models, improving the measurement approach, adding more FLOSS systems to the sample and using more advanced techniques than linear regression.

One of the most interesting findings of our study was that, when removing suspiciously large jumps in the growth trend of FLOSS projects over time (i.e., calculating the net size), simple regression trends can model this 'net' growth very well. In agreement with previous studies [9], these trend models revealed either linear, sublinear or superlinear growth (depending on the system studied). These regular growth trends could be a surprise to FLOSS community leaders and stakeholders and may motivate them to quantitatively examine the evolution of their projects.

We showed how our best models could be used to estimate effort, duration and team size for the development (via evolution) of a 1 MLOC system. Since FLOSS projects seek mainly to attract volunteers and not to hire paid professionals, the current practical use of effort models will be very limited. In the future, however, this type of measurements and models may find their way into helping FLOSS communities to achieve their goals in a more satisfactory manner, probably in a yet unforeseen way.

Acknowledgements

We are grateful to Andrea Capiluppi for comments on an early draft of this paper. We thank the anonymous reviewers for their helpful comments. This work has been funded in part by the European Commission, under the FLOSSMETRICS (FP6-IST-5-033547) and QUALOSS (FP6-IST-5-033547) projects, and by the Spanish CICyT, project SobreSalto (TIN2007-66172). The research reported here was carried out in the context of the Action de Recherche Concertée AUWB-08/12-UMH 19 funded by the Ministère de la Communauté française - Direction générale de l'Enseignement non obligatoire et de la Recherche scientifique. We are grateful to the Belgian F.R.S.-F.N.R.S. for funding the work of one co-author (JFR) through postdoctoral scholarship 2.4519.05.

References

1. Amor, J.J., Robles, G., Gonzalez-Barahona, J.M.: Effort estimation by characterizing developer activity. In: EDSER '06: Proceedings of the 2006 international workshop on economics driven software engineering research, pp. 3–6. ACM, New York, NY, USA (2006)
2. Bellon, S., Koschke, R., Antoniol, G., Krinke, J., Merlo, E.M.: Comparison and evaluation of clone detection tools. IEEE Trans. Software Engineering pp. 577–591 (2007)
3. Boehm, B.: Software Engineering Economics. Prentice Hall (1981)
4. Boehm, B.W., Abts, C., Brown, A.W., Chulani, S., Clark, B.K., Horowitz, E., Madachy, R., Reifer, D.J., Steece, B.: Software Cost Estimation with COCOMO II. Prentice Hall (2000)

5. Concas, G., Marchesi, M., Pinna, S., Serra, N.: Power-laws in a large object-oriented software system. IEEE Trans. Software Engineering **33**(10), 687–708 (2007)
6. Feitelson, D.G., Heller, G.Z., Schach, S.R.: An empirically-based criterion for determining the success of an open-source project. In: Proc. Australian Software Engineering Conf. (ASWEC), p. 6 pp. (2006)
7. Fernandez-Ramil, J., Izquierdo-Cortazar, D., Mens, T.: Relationship between size, effort, duration and number of contributors in large floss projects. In: Proc. of BENEVOL 2008, 7th BElgian-NEtherlands software eVOLution workshop, Technical Report. Eindhoven University of Technology, Eindhoven, The Netherlands (2008)
8. Fernandez-Ramil, J., Lozano, A., Wermelinger, M., Capiluppi, A.: Empirical studies of open source evolution. In: T. Mens, S. Demeyer (eds.) Software Evolution, pp. 263–288. Springer-Verlag (2008)
9. Herraiz, I., Robles, G., Gonzalez-Barahona, J.M., Capiluppi, A., Ramil, J.F.: Comparison between SLOCs and number of files as size metrics for software evolution analysis. In: Proc. European Conf. Software Maintenance and Reengineering (CSMR), pp. 206–213. Bari, Italy (2006)
10. Izquierdo-Cortazar, D., Robles, G., Ortega, F., Gonzalez-Barahona, J.: Using software archaelogy to measure knowledge loss in software projects due to developer turnover. In: Proceedings of the Hawaii International Conference on System Sciences (HICSS-42). Hawaii, USA (2009)
11. Koch, S.: Open Source Development, Adoption and Innovation, *IFIP International Federation for Information Processing*, vol. 234, chap. Exploring the Effects of Coordination and Communication Tools on the Efficiency of Open Source Projects using Data Envelopment Analysis, pp. 97 – 108. Springer Boston (2007)
12. Koch, S.: Effort modelling and programmer participation in open source software projects. Information Economics and Policy **20**(4), 345–355 (2008)
13. Koch, S.: Exploring the effects of SourceForge.net coordination and communication tools on the efficiency of open source projects using data envelopment analysis. Empirical Software Engineering (2009). (forthcoming, DOI: 10.1007/s10664-008-9086-4)
14. Lawrence, K.D., Arthur, J.L.: Robust Regression: Analysis and Applications. CRC Press (1990)
15. Lehman, M.M., Belady, L.A. (eds.): Program Evolution: Processes of Software Change. Apic Studies In Data Processing. Academic Press (1985). (electronically available at http://w3.umh.ac.be/evol/publications/books.html)
16. Lehman, M.M., Ramil, J.F., Sandler, U.: An approach to modelling long-term growth trends in software systems. In: Proc. Int'l Conf. Software Maintenance (ICSM), pp. 219–228 (2001). DOI 10.1109/ICSM.2001.972735
17. Mockus, A., Fielding, R.T., Herbsleb, J.D.: Two case studies of Open Source software development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology **11**(3), 309–346 (2002)
18. Molokken, K., Jorgensen, M.: A review of surveys of software effort estimation. In: ISESE '03: Proceedings of the 2003 International Symposium on Empirical Software Engineering, pp. 223 –230. IEEE Computer Society, Washington, DC, USA (2003)
19. Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S., Lakhani, K.: Understanding free/open source software development processes. Software Process: Improvement and Practice **11**(2), 95 –105 (2006)
20. Van Rysselberghe, F., Rieger, M., Demeyer, S.: Detecting move operations in versioning information. In: Proc. European Conf. Software Maintenance and Reengineering (CSMR), pp. 271–278. IEEE Computer Society Press (2006)

21. Wolverton, R.W.: The cost of developing large-scale software. IEEE Trans. Computers **C-23**(6), 615 – 636 (1974)
22. Xing, Z., Stroulia, E.: Refactoring practice: How it is and how it should be supported - an eclipse case study. In: Proc. Int'l Conf. Software Maintenance (ICSM), pp. 458–468. IEEE Computer Society Press (2006)
23. Yu, L.: Indirectly predicting the maintenance effort of open-source software. Journal of Software Maintenance and Evolution: Research and Practice **18**(5), 311–332 (2006)