Free/Libre and Open Source Software Metrics

Sponsored through Framework Programme Sixth (Call 5) by

Document Information

Version: 1.0
Date : Nov 15, 2009
revision: 1

Owning Partner: WUW

Author(s):
Stefan Koch
Santiago Dueñas
Israel Herraiz
Daniel Izquierdo
Ioannis Stamelos
Ioannis Samoladas
Sulayman K. Sowe
Kirsten Haaland
Rishab Ghosh

Reviewer(s):
Jesus M. Gonzalez-Barahona

To:
CONSORTIUM

Purpose of distribution:
Final copy

The FLOSSMetrics Consortium consists of: Universidad Rey Juan Carlos, University of Maastrich, Wirtschaftsuniversitaet Wien, Aristotle University of Thessaloniki, Conecta s.r.l., Zea Partners and Philips Medical Systems PMS Nederland B.V.

Printed on at

Status:		**Confidentiality:**		
[ ]	Draft	[ ]	Public	- Intended for public use
[ ]	To be reviewed	[ ]	Restricted	- Intended for FLOSSMETRICS consortium only
[ ]	Proposal	[x]	Confidential	- Intended for individual partner only
[x]	Final/Released			

Deliverable ID: **D5.1-A**

Title:

High Level Studies - Annex

(not for redistribution)

Deliverable: D5.1-A

Title: High Level Studies - Annex

Executive Summary:

This report details the focussed studies which were undertaken using the FLOSSMetrics database and data set. They do not intend to conform a comprehensive analysis of FLOSS development, but a set of examples of how the data collected by FLOSSMetrics can be used for research studies. All the studies performed have in common some context related to software development. However, they make use of different sources of information, and in some cases they aggregating results from several of those different sources. For each study, a standardised format is used for a quick description, in order to ease accessibility and comparisons. Whenever possible, in-depth information is also provided for the studies, especially on context and motivation, as well as methodology and preliminary results.

Note: This document is the annex to the release of the High Level Studies report. Due to copyright issues the redistribution is forbidden.

1. APPENDIXES

This section includes research articles based on the studies described in the main document. The idea is to provide some results and useful information about to the researches about the possibilities of FLOSSMetrics database.

The list of papers is the following:

- "Characterization of the evolution of software" (Study: Characterization of the Evolution Dynamics of Software)
- "Evolution of the core team of developers in libre software projects" (Study: Evolution of Core Team Members)
- "What Does It Take to Develop a Million Lines of Open Source Code? " (Study: Effort)
- Assessing FLOSS Communities. An Experience Report from the QualOSS Project (Study: Quality in Open Source Metrics)
- FLOSS Communities: Analysing Evolvability and Trustworthiness from an Industrial Perspective (Study: Quality in Open Source Metrics)
- "A study of the properties of libre software" (Study: Correlation Size and Complexity Metrics)
- "Are FLOSS developers committing to CVS/SVN as much as they are talking in mailing lists? Challenges for Integrating data from Multiple Repositories" (Study: Development vs Communication)
- "On the approximation of the substitution costs for free/libre open source software" (Study: Substitution Cost Estimation)
- Survival analysis on the duration of open source projects (Study: Project Survival)

Characterization of the evolution of software

Israel Herraiz
GSyC/Libresoft
Universidad Rey Juan Carlos, Spain
herraiz@gsyc.urjc.es

Abstract

It has been proposed that software evolution follows a Self-Organized Criticality (SOC) dynamics. This fact is supported by the presence of long range correlations in the time series of the number of changes made to the source code over time. Those long range correlations imply that the current state of the project was determined time ago. In other words, the evolution of the software project is governed by a sort of determinism. But this idea seems to contradict intuition. To explore this apparent contradiction, I have performed an empirical study on a sample of 78 libre (free, open source) software projects available in the datasets of FLOSSMetrics, finding that their evolution projects is short range correlated. This suggests that the dynamics of software evolution may not be SOC, and therefore that the past of a project does not determine its future except for relatively short periods of time, at least for libre software. These conclusions were already found in a paper published in the 2008 edition of the International Working Conference on Mining Software Repositories, and in my PhD thesis, and are now verified with a sample of projects obtained from FLOSSMetrics.

1 Introduction

Libre[1] software development has been traditionally a source of strange cases of software evolution. The first to report one of those were Godfrey and Tu [6, 7]. Their findings suggested that the classical Lehman's *laws of software evolution* [15] were not fulfilled in the case of Linux, because it was evolving at a growing rate (which in fact was still growing 5 years later [20]).

Those cases raised the question of whether libre software evolves differently than propietary software, and whether the *laws of software evolution* are a valid approach for an universal theory of software evolution.

Although these questions have been addressed many times [14], most of the findings and models exposed on those works have failed to provide the theoretical background needed for a proper and universal theory of software evolution. One study that addressed the problem was Wu, in his PhD thesis [25], who among other interesting findings proposed that the evolution of libre software was governed by a Self Organized Criticality (SOC) dynamics.

This conclusion was supported by the presence of long range correlated time series in a set of 11 projects. Regardless the suitability of the selected projects for this kind of study, the limited amount of cases studies, or even the methodology used, we find the idea of long range correlated processes in software evolution as contrary to common intuition. Long range correlation would mean that the current state of the project is determined (or at least, heavily influenced) by events that took place long time ago. In other words, the evolution of libre software is governed by a sort of determinism.

In previous works [10, 8], I already found that evolution was short range correlated for a sample of projects extracted from SF.net, contrarily to the results found by Wu. In this paper, I repeat the methodology with a sample of projects extracted from FLOSSMetrics, to verify whether or not the same short range correlations hold for these cases. As in the mentioned papers, I have studied the daily time series of changes, focusing on deciding whether their profile were short or long range correlated.

The rest of the paper is structured as follows. The next section presents some background on the question of software evolution, and the particular case of libre software evolution. Section three describes the data sources used to extract the sample of projects that have been studied in this work, followed by a description of the methodology used, in section four. In the next section, the results are shown and discussed. The sixth section includes a brief analysis of the sensitivity of the results. The next section includes a summary of the results, to highlight the findings of this

[1] In this paper I will use the term "libre software" to refer both to "free software", as defined by the Free Software Foundation, and "open source software", as defined by the Open Source Initiative.

study. The eighth section discusses some threats to the validity of this study. The ninth section discusses the conclusion based on the empirical findings of this work. Finally, the last section includes some acknowledgments of the research projects that have funded this work.

2 Related work

Software evolution started as field of research thirty years ago, with the seminal work done by Lehman. His aim is still to obtain a theory of software evolution. The *laws of software evolution* (first formulated 1985 [15], and revised during the nineties [17]) were a first step towards this direction. These laws were the base for some evolution models made by Turski [23, 24].

However, both the laws and the assumptions made to obtain the mentioned models, have not been verified in some case studies. The most notable is the case of the Linux kernel [6, 7], that evolves at a growing rate. One of the laws of software evolution is precisely that software evolves at a declining rate because of increasing complexity. That case in particular was labeled by Lehman as an anomaly [16].

The truth is that the quest for a theory of software evolution is still open. In a recent book [18], Lehman and Fernandez-Ramil come over the question of what are the requirements of a theory of software evolution.

Many have tried to obtain a model for software evolution. There are two main approaches that I label as the *statistical* and the *physical* approaches. Among the *physical* approaches, as well as the works by Turski already cited, I mention [1, 4, 21]. These are three different approaches that have tried to model how software evolves based on some theoretical assumptions, and building a predictor model on top of them. However, to my knowledge, no empirical validation at large scale has been done (using those models, or any other similar model).

The *statistical* approach has been more deeply studied. I cite again the works by Godfrey and Tu about the evolution of Linux, or the update of that work by Robles *et al.* [20]. Some studies have tried to use time series analysis (the same approach that I use in this paper). For instance, [13] used ARIMA models to predict the monthly number of changes in a software project (that was the winner model on the 2007 MSR Challenge [9]). The same approach has been used by other authors [2, 3]. Indeed, although time series analysis is not yet the most popular approach in the empirical software research community, it was proposed as early as 1985 [27, 28, 29].

Nowadays, the most popular statistical techniques for empirical studies of software research seem to be regression analysis and principal component analysis (PCA). As an example, besides Godfrey and Robles *et al.*, I cite [5, 14] for studies that use regression analysis and [19] for a study that use PCA to model the evolution of some operating systems.

In this sense, the PhD thesis by Wu [25] uses time series analysis in order to evaluate the kind of dynamics that drive software evolution. His approach is based on the notion of the Hurst exponent, whose value may be used to classify a process as long term or short term correlated. The empirical study is based on 11 libre software projects. The conclusions are that the 11 projects present long term correlations when the time series of the daily number of changes are studied. We recommend to read that thesis for full details on the notion of long term correlated processes, and for all those interested on a possible model for the evolution of libre software.

His approach was also presented in the 2007 edition of the IEEE International Conference on Software Maintenance [26].

For a detailed description of the possibilities of statistical analysis of software evolution, I refer to my PhD thesis [8].

3 Data source

In this study I have considered the evolution of 78 software projects available in the FLOSSMetrics aggregated database. From that database, I have used the activity in the version control system, in particular, the daily number of changes. Originally, the database contains 100 projects, but only 78 of them includes enough information to perform this study: either they are very young (younger than a year), or some statistical information like code size is missing). two projects from the initial sample.

Table 1 shows some statistical properties of the selected projects. The first column corresponds to number of committers that have contributed at least once to the version control repository. Age measures the time difference between the first and the last commit in number of months. The third column corresponds to the size in SLOC [2] of the latest version available in the version control system. The fourth column also indicates size, but in number of source code files (excluding non-source code files such as documentation). The last column is number of commits recorded in the version control system, and it is an indicator of the volume of activity in the projects.

All the values approximately correspond to March 2009 (the exact dates differ from project to project).

4 Methodology

The methodology had three main steps:

1. Data retrieval
   I downloaded the FLOSSMetrics aggregated database,

[2] This is number of lines excluding blank and comment lines.

	# Developers	Age	SLOC	# Files	# Changes
Min.	1	13	4303	10	84
Max.	5,564	179	1,817,569	7,918	134,812
Mean	185	87	281,886	941	11,928
Median	102	90	151,593	458	3,840
Sd. dev.	593	38	332,228	1,229	23,464

Table 1. Statistical properties of the sample of the 78 projects. Age indicates the number of months that have elapsed between the first and the last commits in the control version system repository. SLOC measures size in Source Lines of Code (it excludes blank and comment lines).

and obtained all the data necessary for the analysis from that database.

2. Time series analysis
   After gathering all the changes history, I calculated the daily number of changes for each project. For each project I obtained a time series that required further processing. I had to apply a smoothing procedure to remove some noise from the data. After removing the noise, I calculated the *autocorrelation coefficients*, in order to find out whether the project was a short or long term process.

3. Statistical analysis
   In order to quantify how many of the projects were described as short memory and how many as long memory, I used regression analysis and then analyzed the distribution of these regressions.

Some of these steps are detailed in the following subsections.

4.1 Time series analysis

One of the main properties of time series is the autocorrelation coefficients. These are the linear correlation coefficient among the time series and the series itself but shifted one position to the future. Thus, we may obtain up to $n-1$ coefficients, where n is the number of *lags* of the series. For instance, if the time series was collected daily, each day will be a lag. Figure 1 shows a diagram that explains how the coefficients are calculated. For instance, $r(1)$ is calculated correlating the original series against the series shifted one position, $r(2)$ is the same but the series is shifted two positions. Following this procedure iteratively, we obtain $n-1$ coefficients, being n the number of points of the original time series (also called lags, as mentioned above).

The autocorrelation coefficients measure the linear predictability of the series, using only values of the past of the same series. What is more important, the shape of the plot of the autocorrelation coefficients gives us information about the kind of process that we are studying.

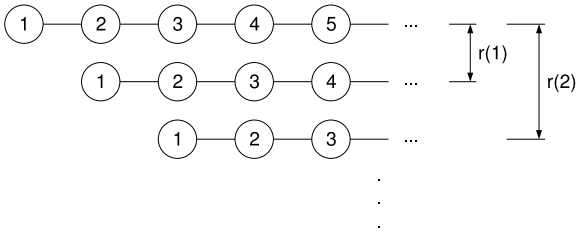


Figure 1. Autocorrelation coefficients in a time series. The coefficients are calculated correlating the series against the same series but shifted one position. If the series has n elements, there are $n-1$ coefficients. In the plot, each circle represents a point in the series. The plot shows how the series is progressively shifted, and how each coefficient is obtaining by linear correlations of the original series and its shiftings.

Short and long term processes present a very different profile. Figure 2 shows the *theoretical* profiles for these processes. The plot is in logarithmic scale. Long term processes present a slow decay of the coefficients. Equation 1 shows the relationship among autocorrelation coefficients $r(k)$ and time lags k for an ideal long term process.

$$r(k) = Ck^{2 \cdot H - 1} \quad k \in [1, n-1] \tag{1}$$

where H is the Hurst exponent (explained with detail in [25]) and n is the number of time lags (or number of points of the time series). k is an integer.

The ideal long term process present a linear profile in the logarithmic plot of autocorrelation coefficients against time lags, the slope of the line may be used to obtain the Hurst exponent. The usual example of a long term process is the Nile River, because the floods come in cycles, and given the data of floods in past years, future floods may be predicted.

In the other hand, short term processes present a fast decay of the coefficients, being the ideal process a linear re-

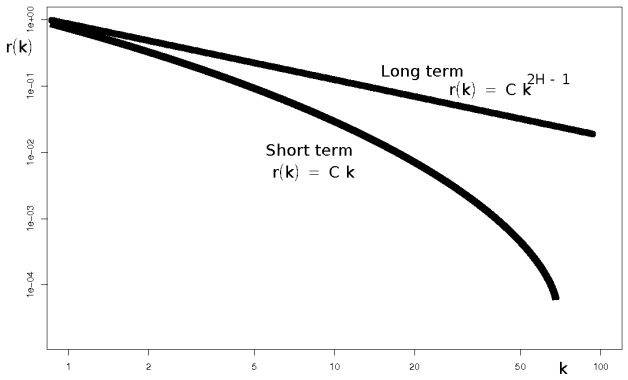


Figure 2. Theoretical profiles of the autocorrelation coefficients of long term and short term processes. This diagram shows the mathematical relationship among the coefficients and the time lags. That equation may be used to obtain the Hurst exponent in the case of long term processes. Note the logarithmic scale of the axis.

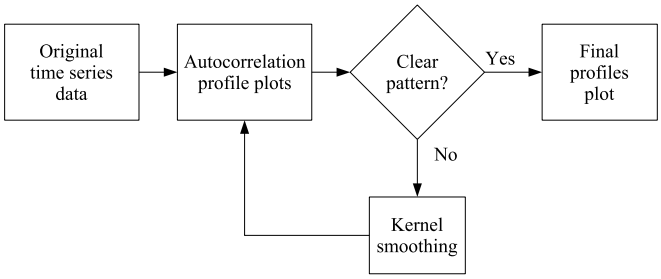


Figure 3. Smoothing and profiles plot process. The original data was very noisy, and smoothing was needed in order to obtain a clear profile.

lationship among the coefficients and the lags. Equation 2 shows the linear relationship among autocorrelation coefficients $r(k)$ and time lags k for an ideal short term process.

$$r(k) = C \cdot k \quad k \in [1, n-1] \tag{2}$$

The typical example of a short memory process is the stock market, where the value of the index today depends at most on values of the index during the last days, but very old results of the index do not affect its current value. In other words, it is very hard to predict values of the stock index far in the future.

A real process would be somewhere among those extreme profiles.

In his PhD thesis [25], Wu gives more details about these same examples of short and long range correlated processes.

In the case of this study, the gathered data was noisy, and the profile of the autocorrelation coefficients was not clear (the values were dispersed, and could be fitted both to a curve and to a straight line). In order to obtain a clean profile, I applied kernel smoothing [22]. The procedure to obtain the profiles is summarized in figure 3. This kernel smoothing filter makes the series smoother by introducing some autocorrelation in the data. If too much smoothing is done, the data will artificially show a pattern due to the autocorrelation added to the data. I discuss the influence of the degree of smoothing on the results in the section devoted to the threads to validity. In any case, I have validated this smoothing procedure in other cases [9, 11, 10, 8], and in those occasions the results were not affected by the smoothing.

When trying to model (or predict) a time series, the kind of process (or profile) is important, because one of the parameters of the model is the *memory* of the process. The memory is the number of points in the past that are influencing the current and future values of the series.

Long term processes present a high memory value, while short term a low value. From my experience [11, 10, 8], software processes (or at least, the study of the evolution of changes in libre software projects) present a memory of no more than 1 week (7 points for daily collected series, as in the case of this study). It is even less in some particular cases. For instance, for Eclipse [9], I think that the memory is no more than 3 days as we presented at the 2007 MSR Prediction Challenge. The goal of the challenge consisted in predicting the number of changes in Eclipse during February, March and April 2007 (submitting obviously the prediction before those months). Using a different model for each component of Eclipse (this is, using a different memory parameter for each module), my approach won the challenge and no component had a memory greater than 3 days [9].

Summarizing, once the autocorrelation coefficients have been obtained, the plot of those coefficients indicates whether the process is short or long term. A regression analysis can help to find out the kind of processes in an automatic way. I describe that step in the next subsection.

4.2 Statistical analysis

After obtaining the time series, and calculating the autocorrelation coefficients, I used regression analysis to find out the kind of profile of each project.

I correlated the autocorrelation coefficients against the time lag, obtaining the Pearson coefficient for each project.

Minimum	.2617
Maximum	1.0000
Mean	.5691
Median	.5648
Sd. dev.	.1457

Table 2. Statistical properties of the set of Pearson correlation coefficients. The values range from very low (0.32, long term process) to very high (0.99, short term process). With only this information, we can not quantify how many projects could be classified in each category (short or long term).

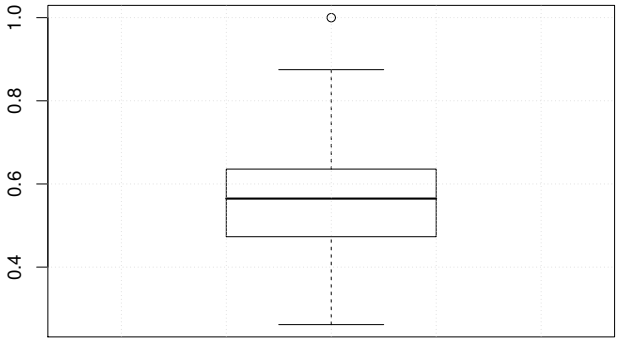


Figure 4. Boxplot of the set of Pearson correlation coefficients.

The Pearson coefficient measures the linear correlation between two variables. The value of the coefficient falls in the interval $[-1, 1]$. The closer its absolute value is to 1, the stronger the linear relationship is. In other words, if two variables do not have a linear relationship, the absolute value of the Pearson coefficient would be much lower than 1.

Therefore, considering equation 2, if the process is close to an ideal short term process, the Pearson coefficient should be close to 1. On the other hand, if the process is not short term, the coefficient should indicate no linear relationship among the parameters.

I repeated the mentioned procedure for the 78 projects. The result was a set 78 Pearson coefficients. As shown in table 2, the values of the coefficients ranged from 0.26 to 1.00. To quantify how many projects could be classified as short term or long term, we estimated the probability density function of the distribution of coefficients, plotted the boxplot and calculated the quantiles of the sample. This is explained in the next section.

5 Results

As mentioned before, I obtained 78 daily time series of number of changes. For each series, we calculated the autocorrelation coefficients $r(k)$ (being k the time lag), and after that calculated the absolute value of the Pearson correlation coefficient of $r(k)$ against k.

This gave me a set of 78 values. Table 2 summarizes the statistical properties of this set. At a first glance, it seems that the values are distributed around average values (this is for instance ~ 0.6), which indicates a weak linear relationship, but still closer to a short term process than to a long term one.

We can represent the same data in a boxplot, in order to obtain a more meaningful description of the data. This boxplot is shown in figure 4, that shows that the sample contains two groups of projects. One group is located around the mean value (~ 0.57), and contains the majority of projects. The other group appears at values around ~ 0.85. This second group clearly corresponds to short range correlated projects. There are no more groups in the data.

Summarizing, the sample contains a group of projects that are clearly short range correlated, and another group with a weak linear relationship, that could correspond to higher memory processes.

An estimation of the density function will help to quantify how many projects are in a certain range of values (like those approximated ranges mentioned in the above paragraph). The estimation of such function is shown in figure 5. As that figure shows, it seems that there is a group of projects with very high values, and another group around a value of approximately 0.85 (this is, a group around the value of the mean or the median). Regarding the low end of the distribution (this would represent long term correlated projects), it seems that those projects are only a minority.

A more accurate statistical tool to quantify the number of projects is the quantiles of the sample. Table 3 contains the quantiles for the Pearson coefficients. For instance, only 40% of the projects present a correlation coefficient lower than $.8178$, and only 20% lower than $.7394$. In other words, 80% of the projects have a coefficient greater than $.7394$.

According to the results shown in that table, and assuming that a Pearson coefficient of 0.6 indicates a linear relationship and therefore a short range correlated process, 70% of the projects are long term correlated processes (probably, with a wide range of memory values), and 30% of the projects are short range correlated processes (distributed across the two groups mentioned above).

6 Sensitivity analysis

In the previous section we have shown that most of the projects are long range correlated processes, and that a certain amount of the sample corresponds to short range cor-

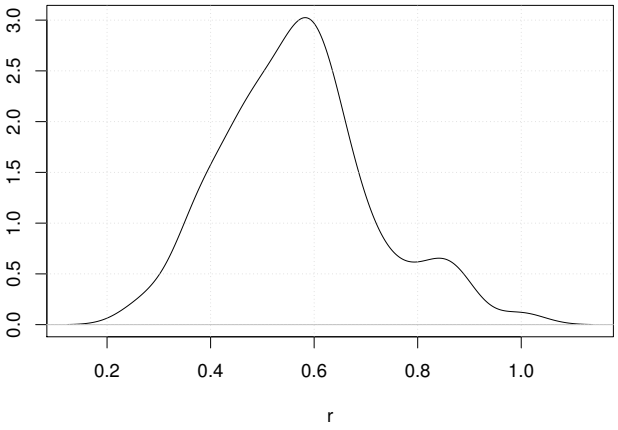


Figure 5. Density function of the set of Pearson correlation coefficients. It seems that there is a group of projects with coefficients very close to 1, and another group simmetrycally distributed around a value of approximately 0.85. Both groups would correspond to short term processes. Long term processes, located in the left tail, are a minority.

Quantile (%)	r
0	.2617
10	.3853
20	.4493
30	.4846
40	.5324
50	.5648
60	.5948
70	.6199
80	.6643
90	.7686
100	1.0000

Table 3. Quantiles of the sample of Pearson correlation coefficients. 70% of the projects present a value lower than 0.6.

related processes. However, those projects are very heterogeneous. For instance, the size of the project varies in a wide range, as the number of developers, age, or any other parameter.

It could happen for instance that the number of developers or the size of the project influences how the project evolves.

In order to find out if the results are sensible to some of the properties of the project, we have performed a brief sensitivity analysis.

I have considered all the factors that are shown in table 1. I have plotted the values of each one of those properties against the value of the Pearson coefficient calculated in the previous section. Thus, we can find if there exists patterns when the projects are clustered in homogeneous groups (for instance, we could find that small projects evolve like short term processes, but large projects do not).

Figure 6 shows the results of the sensitivity analysis. That figure contains six plots. Each plot compares the value of one of the properties shown in table 1 against the value of the Pearson correlation coefficient. Each point correspond to a project. The vertical axis is in logarithmic scale. This is because the kind of distribution of the different properties. For instance, SLOC is distributed following a Pareto-like distribution (there are a few projects that are very large compared to the rest). That kind of data does not show well in linear scale, as only some isolated points appear in a side of the plot, and a set of points grouped in a small area in the other, resulting in a very little meaningful plot. The horizontal axis is in linear scale, and shows the Pearson correlation coefficient. Values of the coefficient close to 1 correspond to short term processes. Values lower than 0.6 may be considered long term processes.

At a first glance, there is no any clear pattern in the data. In other words, in spite of the heterogeneity of the projects, the character of the process (short or long term) is not related to any of the properties.

For instance, let us focus in the case of SLOC. In the range of the Pearson coefficient from 0.5 to 1.0, there small, medium and large projects. If we focus in any other range of SLOC, we also find projects with a wide range of values of the Pearson coefficient.

This behavior is verified with the rest of properties too.

7 Summary of results

Section 5 has shown the *raw* statistical results. Moreover section 6 has shown how the results are affected by the different statistical properties of the projects. In this section I include a brief summary of the main results.

First of all, after analyzing the shape of the profiles of the time series, the main result is that only 30% of the projects can be considered short term processes. It seems that there are two different groups of short memory projects, that are statistically different. However, we could not identify which were the members of each group. Regarding those projects evolving like long term processes, 70% of the projects in the sample evolve like long term correlated processes. This results is in conflict with previous findings [**?**, 8].

In those results, I have not made any differentiation by size or any other characteristic of the projects. For instance, it could happen that large and small projects present different profiles (long or short term), and hence the kind of

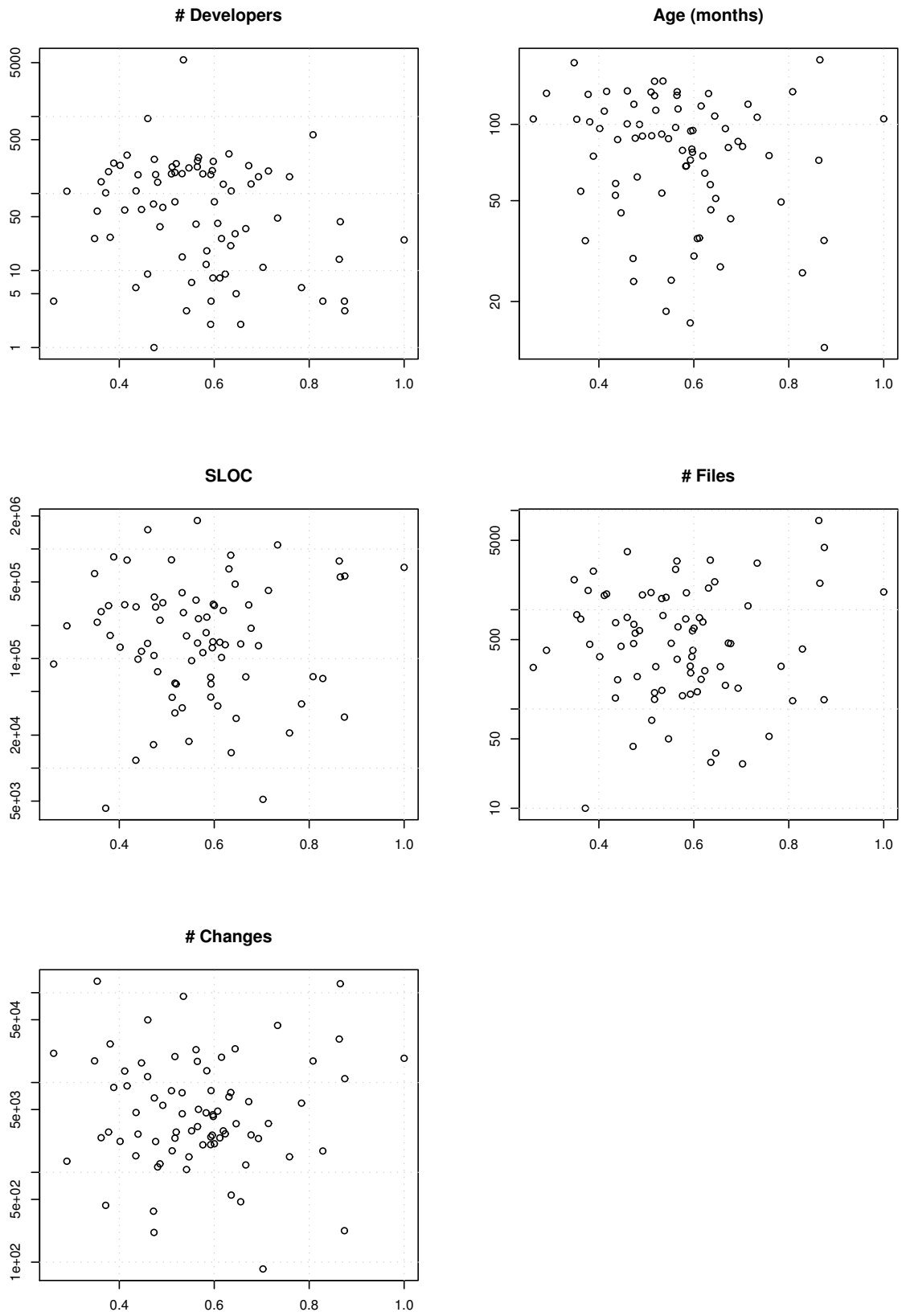


Figure 6. Sensitivity analysis. The plots show the values of each one of the properties shown in table 1 (vertical axis), compared against the Pearson correlation coefficient (horizontal axis). Short term projects present values close to 1. Projects with values lower than 0.6 are long term processes. The vertical axis of some of the plots are in logarithmic scale. The plots show that there is no pattern when dividing the projects in more homogeneous groups. For all the ranges of the Pearson coefficients, there are projects with values of the properties in a wide range.

process could be a stage in the evolution rather than an dynamical property present in the whole life of the project.

To find out if any of the properties that I have measured for the projects (shown in table 1) influenced the results, I plotted the Pearson coefficients against each one of the properties. As discussed in section 5, the Pearson coefficient of short term processes should be close to 1. I consider long term processes are those with a Pearson coefficient lower than 0.6. The plots are shown in figure 6 and discussed in section 6.

The results show that we can find short and long term correlated projects in any size range (measured by number of developers, SLOC, or files), in any age range and in any activity range (measured in number of changes).

8 Threats to validity

I have two main concerns that may affect the validity of our results. The first concern regards the smoothing procedure applied to the time series. This procedure adds some internal autocorrelation, in order to remove some noise. This makes the series *smoother*, and makes it easier to identify a clear profile when plotting the autocorrelation coefficients. However, if too much smoothing is done, the internal autocorrelation may hide the real profile of the data. We are not yet sure about how this smoothing may affect the shape of the profile. I have done some tests for the whole sample, and have not observed any modification of the profile (except to make it clearer) neither of the statistical results shown in this paper. I have used the same procedure in previous studies [9, 11, 10, 8], where the results were compared against real data, and in spite of using smoothing, the time series performed well. Therefore I do not think that the smoothing procedure has changed the profile of the series under study.

The other main concern regards the source for the selected projects. At this moment, the aggregated database of FLOSSMetrics contains very few projects, and from that set, only 78 fulfilled the requirements needed to this analysis. If those projects have not been randomly selected by FLOSSMetrics, the sample could be biased. That could be an explanation of the conflicting results with previous studies that I have done with other samples.

9 Conclusions and further work

Software evolution still lacks a theory that explains how projects are governed over their history. The classical approach by Lehman, summarized in the set of the *laws of software evolution* has found some exceptions, many of them in the libre software community.

The rising of these exceptions has fostered the research on software evolution and libre software, and many models have appeared. Some of them are based on statistical considerations and some others on theoretical assumptions. To date, only statistical models have shown to perform well when compared against real data; however those models can not provide an explanation of the processes that are predicting. One proof of this activity on the software evolution modelling is the annual MSR Challenge, that proposes to predict the evolution of some selected case study.

One of the statistical approaches to the quest of a theory of software evolution is the proposal of a Self-Organized Criticality (SOC) dynamics for software evolution (bounded to the case of libre software). The original proposal [25, 26] seems to fit well with the concepts of libre software development. However, in my opinion, it has a very important drawback: it is not intuitive, as it states that software projects evolve like long term correlated processes. In other words, the current situation was determined time ago: the software project is driven by a sort of determinism.

In order to find if this behavior was common in libre software projects, I have studied the evolution of a sample of 78 case studies. The results show that 70% of the projects evolve like a long range correlated process, and therefore might be explained by a SOC dynamics. 30% of the projects are evolving like short memory processes though. This quantities are in conflict with previous studies, although I have some concerns about the sample being biased.

Furthermore, this study is an example of the possibilities of making large scale empirical research when databases are documented and publicly available. FLOSSMetrics [12] is an example of this kind of collaborative research. The databases contain metrics and facts about thousands of libre software projects (although at the moment of writing this, they are not yet available in an aggregated manner).

Finally, the findings are in conflict with previous results that suggested that SOC may not be a good model for a hypothetical theory of software evolution. This is a very interesting finding, that should stimulate a verification of this study once the full FLOSSMetrics aggregated databases become available.

References

[1] I. Antoniades, I. Samoladas, I. Stamelos, L. Aggelis, and G. L. Bleris. Dynamical simulation models of the Open Source development process. In S. Koch, editor, *Free/Open Source Software Development*, pages 174–202. Idea Group Publishing, Hershey, PA, 2004.

[2] G. Antoniol, G. Casazza, M. D. Penta, and E. Merlo. Modeling clones evolution through time series. In

Proceedings of the International Conference on Software Maintenance, 2001.

[3] F. Caprio, G. Casazza, M. D. Penta, and U. Villano. Measuring and predicting the Linux kernel evolution. In *Proceedings of the International Workshop of Empirical Studies on Software Maintenance*, Florence, Italy, 2001.

[4] J.-M. Dalle and P. A. David. The allocation of software development resources in Open Source production mode. Technical report, SIEPR Policy paper No. 02-027, SIEPR, Stanford, USA, 2003. http://siepr.stanford.edu/papers/pdf/02-27.pdf.

[5] A. R. Fasolino, D. Natale, A. Poli, and A. Alberigi-Quaranta. Metrics in the development and maintenance of software: an application in a large scale environment. *Journal of Software Maintence: Research and Practice*, 12:343–355, 2000.

[6] M. Godfrey and Q. Tu. Evolution in Open Source software: A case study. In *Proceedings of the International Conference on Software Maintenance*, pages 131–142, San Jose, California, 2000.

[7] M. Godfrey and Q. Tu. Growth, evolution, and structural change in open source software. In *Internation Workshop on Principles of Software Evolution*, Vienna, Austria, September 2001.

[8] I. Herraiz. *A statistical examination of the evolution and properties of libre software*. PhD thesis, Universidad Rey Juan Carlos, 2008. http://purl.org/net/who/iht/phd.

[9] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles. Forecasting the number of changes in Eclipse using time series analysis. In *International Workshop on Mining Software Repositories*. IEEE Computer Society, 2007.

[10] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles. Determinism and evolution. In *Proceedings of the 5th International Working Conference on Mining Software Repositories (MSR)*, pages 1–10. ACM, 2008.

[11] I. Herraiz, J. M. Gonzalez-Barahona, G. Robles, and D. M. German. On the prediction of the evolution of libre software projects. In *IEEE International Conference on Software Maintenance*, pages 405–414. IEEE Computer Society, 2007.

[12] I. Herraiz, D. Izquierdo-Cortazar, F. Rivas-Hernandez, J. M. Gonzalez-Barahona, G. Robles, S. D. nas Dominguez, C. Garcia-Campos, J. F. Gato, and L. Tovar. FLOSSMetrics: Free / libre / open source software metrics. In *Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR)*. IEEE Computer Society, 2009.

[13] C. F. Kemerer and S. Slaughter. An empirical approach to studying software evolution. *IEEE Transactions on Software Engineering*, 25(4):493–509, 1999.

[14] S. Koch. Evolution of Open Source Software systems - a large-scale investigation. In *Proceedings of the 1st International Conference on Open Source Systems*, Genova, Italy, July 2005.

[15] M. M. Lehman and L. A. Belady, editors. *Program Evolution. Processes of Software Change.* Academic Press Inc., 1985.

[16] M. M. Lehman, J. F. Ramil, and U. Sandler. An approach to modelling long-term growth trends in software systems. In *Internation Conference on Software Maintenance*, pages 219–228, Florence, Italy, November 2001.

[17] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski. Metrics and laws of software evolution - the nineties view. In *METRICS '97: Proceedings of the 4th International Symposium on Software Metrics*, page 20, nov 1997.

[18] N. H. Madhavji, J. Fernandez-Ramil, and D. E. Perry, editors. *Software Evolution and Feedback. Theory and Practice*. Wiley, 2006.

[19] Y. Peng, F. Li, and A. Mili. Modeling the evolution of operating systems: An empirical study. *The Journal of Systems and Software*, 80(1):1–15, 2007.

[20] G. Robles, J. J. Amor, J. M. Gonzalez-Barahona, and I. Herraiz. Evolution and growth in large libre software projects. In *Proceedings of the International Workshop on Principles in Software Evolution*, pages 165–174, Lisbon, Portugal, September 2005.

[21] G. Robles, J. J. Merelo, and J. M. Gonzalez-Barahona. Self-organized development in libre software: a model based on the stigmergy concept. In *Proceedings of the 6th International Workshop on Software Process Simulation and Modeling (ProSim 2005)*, St.Louis, Missouri, USA, May 2005.

[22] R. H. Shumway and D. S. Stoffer. *Time Series Analysis and Applications. With R Examples*. Springer Texts in Statistics. Springer, 2006.

[23] W. M. Turski. Reference model for smooth growth of software systems. *IEEE Transactions on Software Engineering*, 22(8):599–600, 1996.

[24] W. M. Turski. The reference model for smooth growth of software systems revisited. *IEEE Transactions on Software Engineering*, 28(8):814–815, 2002.

[25] J. Wu. *Open Source Software evolution and its dynamics*. PhD thesis, University of Waterloo, 2006.

[26] J. Wu, R. Holt, and A. E. Hassan. Empirical evidence for SOC dynamics in software evolution. In *IEEE International Conference on Software Maintenance*, pages 244–254. IEEE Computer Society, 2007.

[27] C. C. H. Yuen. An empirical approach to the study of errors in large software under maintenance. In *Proceedings of the International Conference on Software Maintenance*, 1985.

[28] C. C. H. Yuen. A statistical rationale for evolution dynamics concepts. In *Proceedings of the International Conference on Software Maintenance*, 1987.

[29] C. C. H. Yuen. On analyzing maintenance process data at the global and detailed levels. In *Proceedings of the International Conference on Software Maintenance*, pages 248–255, 1988.

Evolution of the core team of developers in libre software projects

Gregorio Robles, Jesus M. Gonzalez-Barahona, Israel Herraiz
GSyC/LibreSoft, Universidad Rey Juan Carlos (Madrid, Spain)
{*grex,jgb,herraiz*}*@gsyc.urjc.es*

Abstract

In many libre (free, open source) software projects, most of the development is performed by a relatively small number of persons, the "core team". The stability and permanence of this group of most active developers is of great importance for the evolution and sustainability of the project. In this paper we propose a quantitative methodology to study the evolution of core teams by analyzing information from source code management repositories. The most active developers in different periods are identified, and their activity is calculated over time, looking for core team evolution patterns. Several activity plots and parameters for characterizing these patterns are presented, and applied to several large, well-known libre software projects, where their effectiveness is validated and discussed.

1. Introduction

Employee turnover is known to be high in the traditional software industry since many years ago [1]. However, in libre software[1] projects the study of developer turnover has not been an active research topic. Most of the attention in this area has been focused on the organizational structure of the projects [2], with little attention to the dynamics of the developers.

A noteworthy contribution in this sense, although it does not address the evolution of developer communities, is the *onion model* [3], which shows how developers and users are positioned in communities. In this model, it is possible to differentiate among core developers (those who have a high involvement in the project), codevelopers (with specific but frequent contributions), active users (contributing only occasionally) and passive users [4], [5].

The onion model provides only a static picture of a project, lacking the time dimension that is required for studying joining and leaving processes. Jensen et al. [6] have studied and modeled the processes of role migration for some libre software communities, focusing on end-users who become developers. This has lead to the finding of different paths for the joining process, concluding that the organizational structure of the studied projects is highly dynamic in comparison to traditional software development organizations. With respect to abandonment, the number of developers leaving a project has been studied in [7] by using the half-life parameter, defined as the time required for a certain group of contributors to fall to half of its initial population; in the case of the Debian project the obtained half-life was of 7.5 years.

Given these precedents, the authors of this paper begun to study projects to better understand the evolution of the group of developers contributing to a libre software project, in particular of the the most active ones. For this, a characterization is proposed based on two extreme scenarios: the "code gods" scenario, in which the composition of the core team is highly stable over time, and the "series of generations" scenario, where several generations succeed each other. In the first case, a project relies heavily on a small number of long contributing developers, which would imply a great risk in case a significant fraction of them leave. In the second case the project shows a series of partially overlapping core teams, with some of the initial members leaving the project, but others joining and filling the gap.

A specific methodology has been designed to quantitatively characterize a project in the spectrum between these two scenarios, and to visualize more in detail the evolution of the core team. The first steps of this methodology, applied to a few projects, are depicted in [8]. An extension and refinement of the methodology is presented in this paper, where it is used on a larger amount of libre software projects.

The rest of this paper is organized as follows. The next section describes the methodology that has been designed to extract data from source code management systems and produce indexes, graphs and maps that help to understand the evolution of the core team. After it, several cases are discussed to illustrate the use of the methodology, and to some extent validate it. These cases are a part of the study of 19 large libre software projects, which are also discussed in combination. A final section with some conclusions and hints about further research closes the paper.

2. Methodology

The methodology used in this study is based on retrieving data about the activity of developers from source code management repositories, which are mined using CVSAnalY [9]. This tool retrieves information about every commit to the

1. In this paper we will use the term "libre software" to refer both to software licensed under terms compliant with the FSF definition of "free software" or with the OSI definition of "open source software".

repository, and inserts it into a database where it can be conveniently analyzed.

2.1. Summary of the methodology

To characterize the evolution of the core team, first the life of the project is split in periods of equal duration. Then for every period i, the most active developers are identified as $CoreTeam_i$. This is done by calculating the number of commits during that period for the most active developers. For each $CoreTeam_i$, its activity is tracked for the rest of the life of the project (before and after period i). Hence, for each period j, the number of commits is calculated for all the developers in $CoreTeam_i$. Finally, the resulting data (that represents the activity of the each $CoreTeam_i$ for all periods) is plotted in several formats, and collapsed into some indexes that allow comparison and classification.

In all the cases, only commits on source code files have been considered, since the study is focused on the behavior of developers working on source code (for further details, please refer to [10]).

There are some cases which can be a source of problems when applying the methodology: both automatic commiters (present in some projects for routine operations) and different patterns of work may influence the results for specific projects. However, experience after analyzing a large quantity of projects tends to show that these effects can, in general, be neglected.

In addition, $CoreTeam_i$ and $CoreTeam_j$ may include developers in common, as a developer can be a part of the most active group in several periods. This is important to notice to correctly interpret the information provided by the methodology.

2.2. Duration of the periods

The selection of the duration of the periods in which the life of the project is divided is basic for the definition of the notion of "history" of a core team. The smaller the period, the more transient effects can be found (such as developers on vacation, or different activity patterns when the project is close to a release). But the larger the period, the lesser detail is captured, maybe losing important transitions, meaningful for our study.

In addition, projects of different life lengths will have different number of periods, if periods of constant duration are used. This could cause difficulties when comparing results, but will at the same time allow to better understand the behaviors related to the length of the history of projects. The opposite option, using a constant number of periods for projects of any length, can be more useful for comparisons, but will neglect those aspects related to project age.

Because of all these trade-offs, we have not considered a single time span for periods. For the purposes of the study, usually the most significant results are obtained by dividing the history of the project into 10 or 20 periods. These also produce periods of a reasonable duration both for relatively young (about 2 to 4 months for projects with 3 years of development) and old projects (6 to 12 months for projects with around 10 years). To ensure that we are not losing any important information, we have obtained several plots using equal time intervals of three months; as it will be shown in later sections, using the former or the later does not give significant differences.

2.3. Identification of the core team

Libre software projects usually show power law or similar distributions for the number of contributions by developers [4], [11], [12]. In other words, a small fraction of all the developers are responsible for a large fraction of all the activity. Because of that, the criterion chosen for the identification of core teams we have chosen is the fraction of developers who produce more commits.

After considering several alternatives, we have found that fractions of 0.1 and 0.2 (that is, the top 10% and 20%) are large enough to capture developers producing most of the activity (usually more than 50%, reaching in many cases as much as 90% or 95% of the total number of commits).

3. Outputs of the methodology

Our methodology provides both some graphs that help to visualize the results and some data (in the form of arrays and indexes).

3.1. Arrays and indexes

The main output of the methodology is the $AbsoluteMatrix$: a squared two dimensional array, with the number of periods as range. Values for each position x, y in the array are the absolute number of commits for $CoreTeam_x$ in period y. Therefore, positions in the diagonal (where $x = y$) correspond to the activity of each core team during the period in which it is actually the core team. Positions where $y > x$ represent the activity of that core team in periods after that moment, while $y < x$ represent the 'past' activity of that team.

From other point of view, each row (same x value) represents the history of activity of $CoreGroup_x$. Each column (same y value) gives the activity of all core groups during period y. These clues can help to interpret absolute matrices, and the graphs produced when plotting their contents.

Absolute matrices, carrying absolute number of commits in each position, are sensitive to the total activity of the project. In some cases, this makes it difficult to compare periods of different levels of activity (for instance, when the focus is the importance of a certain core team during all

the periods). For these cases, the $NormalizedMatrix$ is produced. It is calculated from the absolute matrix, using for each position its original value divided by the total number of commits in the corresponding period:

$$NormalizedMatrix_{x,y} = \frac{AbstoluteMatrix_{x,y}}{TotalCommits_y}$$

An alternative chance for normalization is to use the activity of the core team of each period instead of the total activity in each period (it can be easily shown how the positions in the diagonal in this matrix have always a value of 1):

$$CoredMatrix_{x,y} = \frac{AbstoluteMatrix_{x,y}}{AbsoluteMatrix_{y,y}}$$

Complete arrays provide a lot of information, but they are also in some cases too detailed and difficult to interpret. Therefore, a single parameter, $Index$, which summarizes the information in a matrix, could be calculated as follows:

$$Index = 100 * \sum_{x \neq y} CoredMatrix_{x,y}$$

Note that high values for this index are indicative for a higher "load" on all positions, which means that the activity of the different core teams is high over their whole history (a situation that is close to the "code gods" scenario, as this happens when the composition of the core group changes seldom). The smaller $Index$ is, the more positions with little activity, pointing out the existence of a heterogeneity of developers composing the core teams (having a "series of generations" scenario).

3.2. Graphs

Several graphs are produced to visualize and help with the interpretation of the previous data:

- Absolute graph. Displays the absolute number of commits for each core group (Y axis) for each interval over time (X axis). This graph is obtained by plotting the data in each $AbsoluteMatrix_x$ row.
- Aggregated graph. Displays the aggregated number of commits for each core group since the beginning of the project (Y axis) versus time (X axis). This graph shows the integral of the absolute graph.
- Normalized graph. Displays the fraction of the total commits performed by each core group for each interval (Y axis) versus time (X axis). This graph shows the same information than the absolute graph, but normalized by the total number of commits performed in each period, and can be obtained by plotting the data in each $NormalizedMatrix_x$ row.
- Heat map. Displays the $CoredMatrix$, with a color (or gray-scale) for each position. Provides a quick yet detailed view of the evolution of core groups over time. The history of each core group can easily be observed, with high activity periods as *hot* areas, while periods with little activity will appear *cold*.
- Normalized 3D map. This is a three dimensional view of the $NormalizedMatrix$, with Z axis representing the normalized activity per position. Provides similar information to the one in the heat map, but can be interpreted with more detail if a 3D browser is available.
- Absolute 3D map. This is a three dimensional view of the $AbsoluteMatrix$, with Z axis representing the activity per position. Similar to the normalized 3D map, but information about the total level of activity for each position is also provided.

The combined observation of these graphs, for different time periods (10 or 20), and using different fractions of developers for identifying core groups (top 10% or 20%), provides a complete landscape of the activity of the core group over time project.

4. Some case studies

The methodology described in the previous section has been applied to 19 different projects. All of them are at least six years old, so that they have enough history to analyze. Table 1 in the Appendix shows a summary of the main parameters of these projects (including several indexes, which will be discussed below and are a good estimator for the behavior of their core teams).

Among them, we have selected three cases to illustrate the use of the methodology more in detail. Two of them are close to the extreme scenarios: code gods (the GIMP) and series of generations (Mozilla). The third project (Evolution) is between these to ends, and shows some peculiarities that are worth mentioning. Results for the rest of the case studies are given in combination in the last subsection.

4.1. Case study: the GIMP

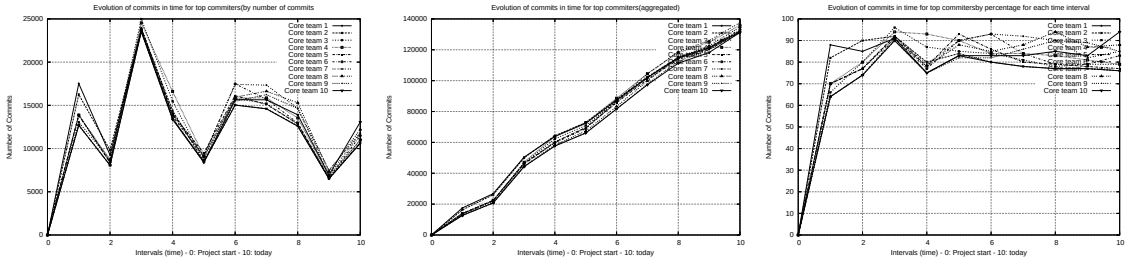
The GIMP can be considered as a canonical example of a project with "code gods". It is a very active project (by number of commits) with many developers involved.

Graphs in figure 1 show the typical pattern of code gods scenarios. The lines in all graphs are almost overlapping, which means that all the core teams have almost the same composition. However, the core team is not always exactly the same. A detailed study of the developers in the core teams yields that one of the most active developers is present in all of them. The second and third most active developers enter during the third interval (which starts around mid 1999) and stay in the project until today.

The normalized graph, also shown in figure 1, provides further information. By construction, the higher curve in each period corresponds to the core team that has been identified in it. In the case of a "code gods" project, the other core groups should be near that maximum (or at the

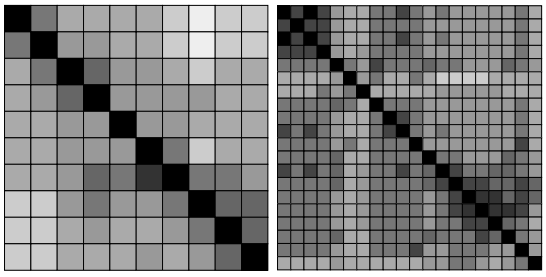
Project	Size	Commits	Commiters	Age	Index			
					10 / 10	10 / 20	20 / 10	20 / 20
Eclipse	4298K	801403	189	68	22.26	33.35	20.22	32.25
Evolution	300K	125938	369	121	32.19	28.79	29.84	29.35
FreeBSD	2085K	241809	345	164	38.61	40.97	36.37	39.48
Galeon	93K	34826	139	120	35.81	28.40	36.52	30.07
GIMP	603K	187522	229	121	45.20	37.22	51.53	39.01
Gnumeric	253K	106327	206	121	49.12	41.14	51.56	46.76
Kdebase	373K	251241	550	117	31.3	33.83	30.78	35.40
Kdelibs	585K	241916	572	117	36.68	34.42	38.51	35.67
Kdenetwork	318K	157762	387	115	20.64	24.68	19.17	22.80
Kdepim	549K	154833	301	108	34.10	33.51	29.22	30.44
KOffice	906K	251823	308	105	30.57	31.24	34.29	32.70
Mcs	1757K	183242	216	67	28.02	34.92	25.72	33.48
Mono	297K	35085	143	66	57.32	48.33	48.4	50.58
Mozilla	3940K	1007370	835	106	18.59	23.49	18.68	50.58
Nautilus	101K	71920	332	121	19.06	19.52	19.57	20.18
NetBSD	2888K	465060	287	164	27.64	40.31	26.24	36.03
OpenBSD	1734K	140213	183	132	43.78	43.25	42.50	42.86
OpenOffice.org	5149K	129209	107	76	31.64	38.91	29.87	36.36
PostgreSQL	381K	90282	28	127	68.67	67.67	73.63	64.73

Table 1. Summary of parameters for the projects analyzed. Size is in SLOC, age (of the repository) is in months. For indexes, 10/20 means: number of periods is 10, core teams identified as top 20% (0.2 fraction) of developers.



(a) Absolute (b) Aggregated (c) Normalized

Figure 1. Graphs for the GIMP project. A fraction of 0.2 was used for identifying core teams.



(a) 10 periods; 0.2 fraction (b) 20 periods; 0.1 fraction

Figure 2. Heat maps for the GIMP project. Fraction provides the fraction of of top developers to identify the core team.

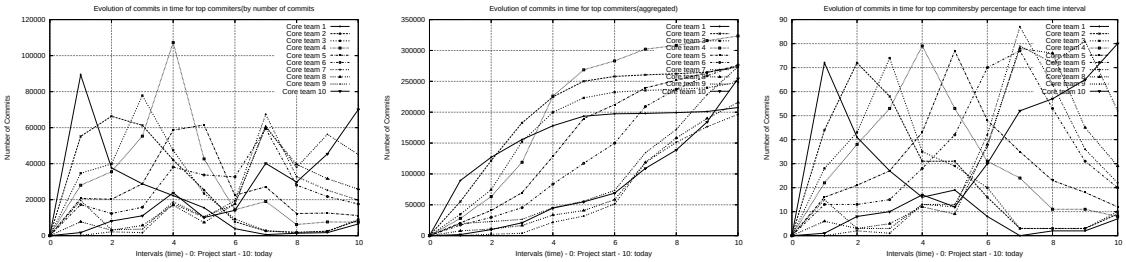
same level if core groups during different time periods have exactly the same composition) as the composition has not changed much over time.

The identification of the code gods scenario is even more evident in the heat maps of figure 2. Except for the diagonal (which is, by construction, always black), the gray color dominates the map, meaning that the composition of the core groups over time is quite similar. The right map, with a higher resolution, shows also the special case of the first core groups: the upper left positions are darker, and are surrounded by lighter ones, showing a change in generations.

The 3D maps of figure 3 provide some more detail. In the normalized map, the lighter plateau that dominates most of the map is a clear indicator of a stable code gods region. Again, the beginning of the project shows a slightly different pattern, with a different composition of the core group.

It is worth noticing that both normalized and absolute 3D maps, when projected on the XZ plane, produce the normalized and absolute graphs. Moreover, thanks to how the normalized map is colored, when projected on the XY plane, the resulting 2D map should result in the heat map. Therefore, these 3D maps in some sense include all the information in the other graphs and maps.

In addition to all this graphical information, the indexes shown in table 1 in the Appendix are also an indicator of a code gods scenarios, being among the highest in the table.



(a) Absolute (b) Aggregated (c) Normalized

Figure 4. Graphs for the Mozilla project. A fraction of 0.2 was used for identifying core teams.

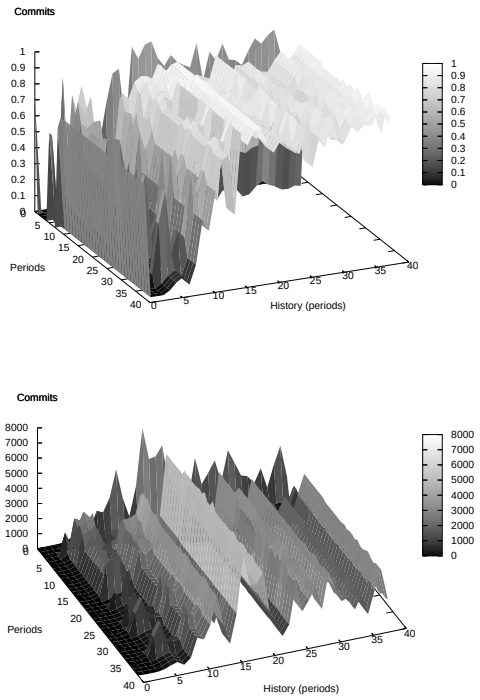


Figure 3. 3D maps for the GIMP project, using quarters as period, and 0.2 as fraction of top developers for identifying core teams. Top is normalized map, bottom is absolute.

4.2. Case study: Mozilla

The Mozilla project is a good example of a series of generations scenario. Figure 4 shows the graphs for the project. Their aspect is, at first sight, clearly different from those of the GIMP. Curves now are very different for each core team. They are not parallel in the aggregated graph, neither almost coincident in the absolute one. In the normalized graph, each team raises from a very small fraction of contributions to a peak of about 80%-90%, and then fades quickly (in about three years) away. In other words, the composition of the core teams varies clearly from period to period. There are clear indicatives of smooth transitions between different core teams: several developers leave or join during each period, but many stay for several periods, ensuring smooth transitions.

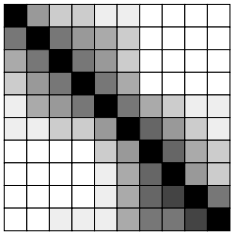


Figure 5. Heat map for the Mozilla project. 10 periods and 0.2 as fraction of top developers identified as core team.

However, even the core teams for the last periods show some activity during the first periods, which means that some very active developers nowadays were already active at those early stages of the project. Correspondingly, the first core team also has activity during the last periods. Therefore, the experience from the beginnings of the project is still available in the core group (at least in the minds of the developers active in core teams during all the life of the project), which is another sign of smooth transitions.

In the heat map (see figure 5) wide areas are white, and most of the activity is concentrated around the diagonal. Again, this shows the succession of generations. Its smoothness can be appreciated but the two or three gray positions around the black diagonal: each core group shows activity before and after its peak. But additional, more subtle information is found in this map. At about periods 5 and 6, a clear transition is observed. Core teams tend to have more activity either before or after those periods. Not surprisingly, this transition happens at 2003, when the corporate support of AOL finished and the Mozilla Foundation was founded. The heat map shows how the composition of teams changed more significantly around that time, although with a certain level of smoothness.

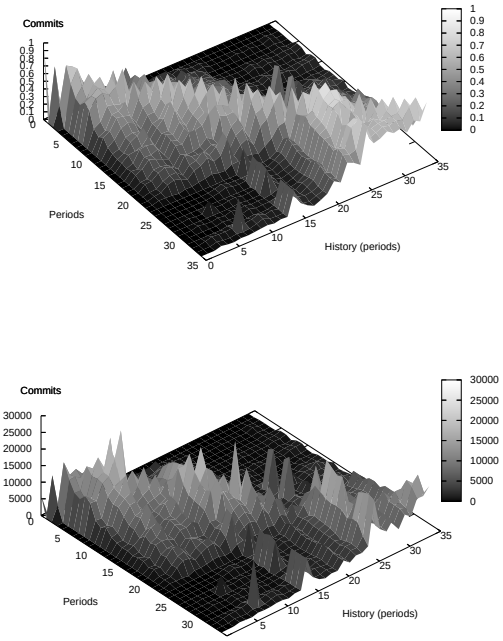


Figure 6. 3D maps for the Mozilla project, using quarters as period, and 0.2 as fraction of top developers for identifying core teams. Top is normalized map, bottom is absolute.

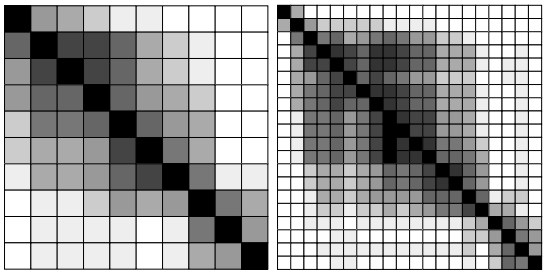
This effect is also apparent in the 3D maps (figure 6, with those wide low, dark areas (that represent low levels of activity for the corresponding core teams). The mountain chain aspect of those maps is a clear indication of changes in the composition of the core teams, which happen ubiquitously for this project. The normalized 3D map is specially illustrative in this respect.

The indexes for this project (offered in table 1) are among the lowest of all the analyzed (except for the 20/20 case).

4.3. Case study: Evolution

Evolution is a good example of a project showing both aspects of the "code gods" and the series of generations scenarios. Evolution started as a community-driven project in December 1998, but was quickly adopted (by the end of 1999) by Ximian (then a small start-up company) as a strategic application. From that point on, Ximian developers had a clear impact on the project. Ximian was acquired by Novell in August 2003, with most of the team working in Evolution also joining the new company.

Both the absolute and aggregated graphs in figure 7 show clearly the low activity of the core team at the beginning of the project, and the impact (in the rise of the activity) of



(a) 10 periods; 0.2 fraction (b) 20 periods; 0.1 fraction

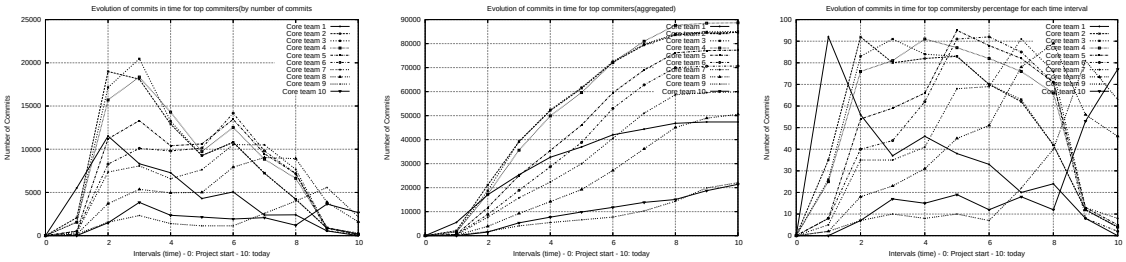
Figure 8. Heat maps for the Evolution project. Fraction provides the fraction of of top developers to identify the core team.

Ximian developers joining the project. At the end of the project, again low levels of activity are found, probably after the spreading of Ximian developers within Novell. These three periods also correspond to differences in the composition of the core team. The normalized graph (also in figure 7) shows how the first core team quickly fades out to less than 50% of activity after less than two years, while the last core teams rise their peak level after periods with very little activity. Between these two ends, the periods in the middle (specially those from 2 to 6, which correspond to the life of Ximian as a company) are close to a code gods scenario (parallel lines in the aggregated graph, almost overlapping curves in the normalized graph). In summary, we can observe an epoch of code gods, while the first and last periods are more close to the series of generations.

Heat maps in figure 8 show the same pattern. The transition from the first periods to the Ximian epoch is smooth but clear. The large square of gray positions (corresponding to 6 core teams in the 10x10 map) points out the code gods period, while the small square in the bottom right suggest the beginning of a new code gods era. The transition to this last square is clearly more sharp than the one at the beginning of the project.

3D maps (see figure 9) are even more clear. The absolute map shows a low elevation at the beginning of the project, and a much higher mountain chain after it. Behind, some small hills correspond to the last periods. The transitions between the three epochs, and the differences in smoothness are obvious.

Finally, the indexes for Evolution are closer to a code gods case than to a series of generations one, which is reasonable if we consider that most of the time, the former is the more similar scenario. However, the aspects related to the transitions are not captured by the indexes (which is not surprising, being the index a really terse summary of a complex pattern).



(a) Absolute (b) Aggregated (c) Normalized

Figure 7. Graphs for the Evolution project. A fraction of 0.2 was used for identifying core teams.

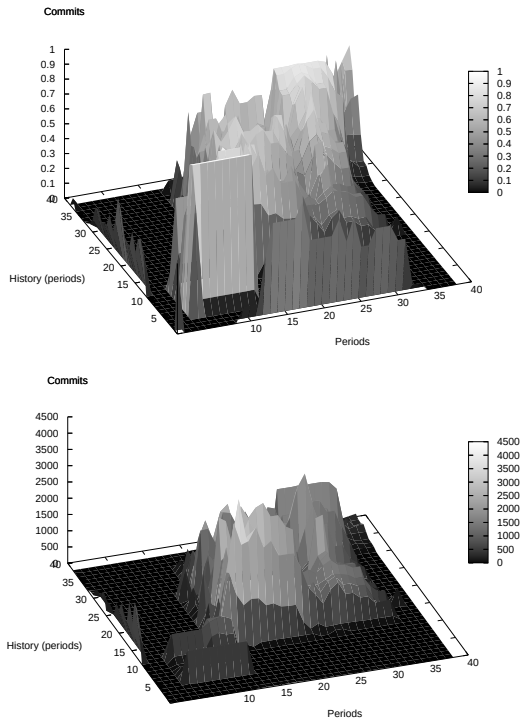


Figure 9. 3D maps for the Evolution project, using quarters as period, and 0.2 as fraction of top developers for identifying core teams. Top is normalized map, bottom is absolute.

4.4. Observations on other projects

After presenting the results of applying the methodology to three projects, some more general observations on the 19 projects analyzed (summarized in table 1 in the appendix) are offered in the following paragraphs.

Especially interesting is figure 10, which plots the indexes of the different projects versus their age. It servers two purposes: to offer a quick graphic summary of the projects, and to show that there is no relationship between age and index.

Showing the lack of correlation between age and index is important, since it could seem intuitive that projects tend to the series of generations scenario with time (given enough time, the chances of developers leaving the project, and others coming in, could be higher). However, we cannot observe this behavior from the studied projects. Those with a larger history in the sample (NetBSD and FreeBSD) show clear differences in their indexes, and in any case they are much more "code gods like" than others with much shorter histories, such as Nautilus or Mozilla. It is also important to notice that for the projects around the 120 months mark, widely varying indexes can be found, ranging from the clear cases of series of generations (Nautilus, Kdenetwork, Mozilla) to those of code gods (OpenBSD, GIMP and Gnumeric). Postgres has (with difference) the highest index. One could argue that this is due to its policy where most commits are performed by a small group of developers, which in many cases commits patches produced by external contributors, but this is really no limitation to the methodology as we are mainly studying if and how this most active group changes over time and not specifically, although also important, its relevance. In this regard, we could state that the methodology is insensitive to *gate keepers* (which is actually a good characteristic as it is a bias that is difficult to minimize).

Some other correlations for the indexes (versus project size, number of developers, number of commits, etc.) have been explored, with no clear result. With the data in our study, it seems that the reasons for the differences in the evolution of the core teams, from project to project, are not related to the characteristics of the software being developed, neither to the level of activity. Other reasons for this differences, maybe linked to the specific policies and procedures of the projects, remain to be found.

Fractional graphs for some of the studied projects are also shown (see table 2 in the appendix). They have been sorted according to their index, from Nautilus, the most close to a series of generations scenario (top left) to Mono and Gnumeric (clear cases of code gods). Each of them is an interesting case, which can be analyzed from the graphs (and the corresponding heat and 3D maps) in detail, as

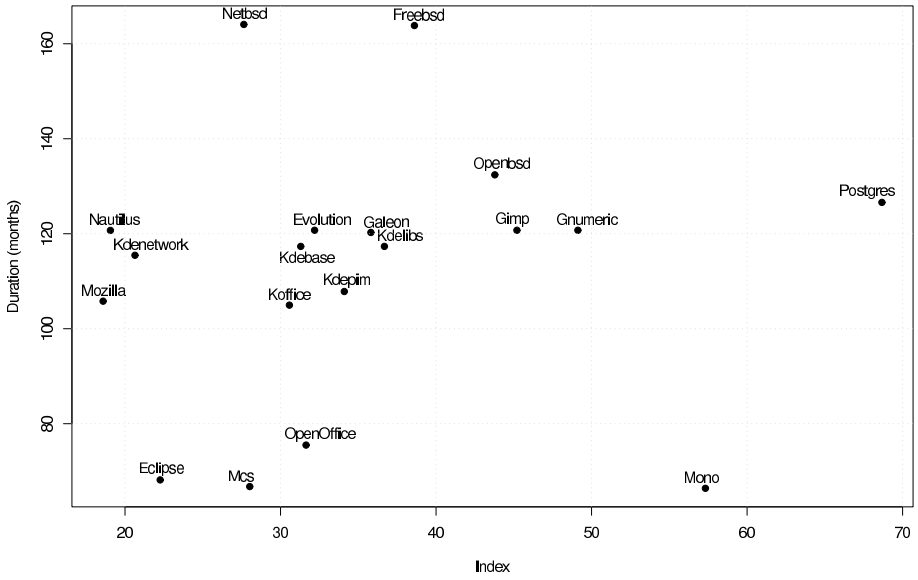


Figure 10. Plot of indexes versus project age, for the projects included in the study. Indexes are calculated by splitting project life in 10 periods, with core teams of top 0.1 of developers.

were Mozilla, GIMP and Evolution. Just as a very brief summary, a single aspect can be highlighted: the differences in smoothness in transitions, which can be inferred from the different rates of ascending and descending curves before and after reaching their peak.

To complete the general discussion of cases, figure 11 shows four plots with different methods for calculating the index, for all the studied projects. Although of course the sample of projects is neither large nor unbiased enough to raise any conclusion about the distribution of indexes, some details can be highlighted. For instance, both plots at the top are quite similar, and the same can be said for the plots at the bottom. Therefore, the index can be considered as almost independent from the fraction of developers considered for identifying the core team (at least if it is between 0.1 and 0.2).

However, the index is less immune to changes in the size of the periods considered: the plots on the top present clear differences with those on the bottom (and a more detailed analysis shows not only differences in the plots, but also in the order of projects). In fact, from an exhaustive analysis of all the projects considered in this study, the indexes calculated with 10 periods match better the real stories of the projects.

5. Conclusions and further research

For the study presented in this paper, a methodology has been designed that allows for a simple yet powerful analysis of the evolution of the core team of libre software projects. The methodology is quantitative, and can be automated, only requiring that the development is performed using a source control management system, and that the researcher has access to the corresponding repository. Fortunately, this is the case for a large fraction of libre software projects, including the most relevant ones.

The methodology can be used to rank projects according to their distance to the two extreme cases of “code gods” and “series of generations”, using the produced indexes. But it provides also a lot of insight on the evolution of the core teams, by showing visually (both in graphs and maps) the activity patterns of the developers forming the core team in each period of the life of a project. This information can be used to identify levels of smoothness in transitions, to detect break points in the evolution of the core team, to understand the differences in activity of the core team in different periods, or to estimate unevenness in the contributions of the most active developers when compared to the rest of them.

In addition, we have applied the methodology to 19 relevant libre software projects, and used these case examples to validate it and illustrate some of its benefits. We also have shown the impact of some events in the history of specific projects on the evolution of their core team (such as the influence of the strategy of companies in the cases of Evolution and Mozilla).

Some factors not specifically discussed in this paper could influence the appropriateness of the methodology. Among them, the relevance of using the number of commits as a proxy for the activity and importance of developers. For validating it, we have studied some other parameters, such as the number of changed lines, without finding meaningful differences. However, an important problem remains open:

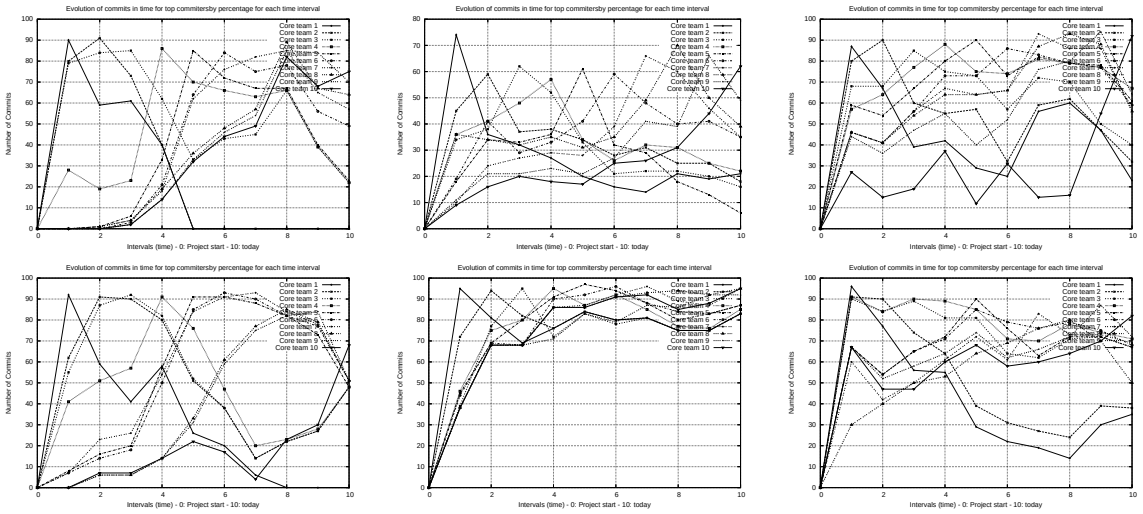


Table 2. 3x2 matrix with fractional generation plots for 6 libre software systems. From left to right and top to bottom, they are Nautilus, Eclipse, KDE base libraries, Galeon, Gnumeric and Mono. Projects closer to the series of generations scenario have been situated at the top, while those with code god patterns are at the bottom.

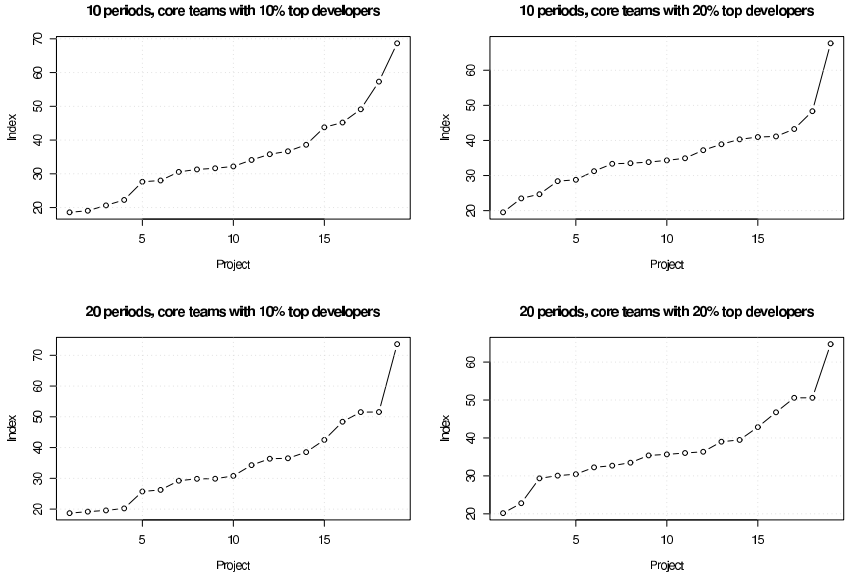


Figure 11. Plots of indexes for all the studied projects, using different numbers of periods and criteria for identifying core teams.

to which extent other, non-coding activities (such as discussion, writing of documentation, or even mediation between developers) should be considered to better identify the core team of developers. This should be the focus of further research.

Another open field for research is the use of the methodology in classical (non-libre) software projects. The fact that many developers in libre software projects are volunteers can provide very interesting information about the natural behavior of programmers, as these developers are self-selected (i.e., there is no traditional, mandatory task assignment as it can be found in the commercial world). In this regard, one of the findings that should be further researched is the amount of time for turnover. From our limited set of projects we have seen that, for those projects with several generations, the time span for a generation ranges from three to five years. This could be indicative for a programmers moving to a different project to keep his motivation and interest on his work high. Having developers enrolled in companies (such as the cases of Mozilla and Evolution) and volunteer

developers in these projects could give further insight to this question in subsequent research.

The sample of projects considered in our study is small, which obviously opens opportunities for validating the methodology with a larger, and more diverse collection of projects. Of course, the larger the projects the more interesting the findings are (since it is difficult to understand all the details of such complex projects without the help of methodologies and tools). But even in the case of small projects some interesting results could be quickly and automatically obtained.

In any case, from our work we can conclude that the study of the behavior of human resources in libre software projects and in software engineering in general, and the relationship between its join/leave patterns and the evolution of the project, is a field worth to explore. This paper tries to be a first step in this direction, focused on studying its dynamics, and on finding how projects cope with the changes caused by it.

Acknowledgment

This work has been funded in part by the European Commission, through projects FLOSSMetrics, FP6-IST-5-033982, QUALOSS, FP6-IST-5-033547, and Qualipso, FP6-IST-034763.

References

[1] B. W. Boehm, Ed., *Software risk management*. Piscataway, NJ, USA: IEEE Press, 1989.

[2] D. M. Germán, "The GNOME project: a case study of open source, global software development," *Journal of Software Process: Improvement and Practice*, vol. 8, no. 4, pp. 201–215, 2004.

[3] K. Crowston and J. Howison, "The social structure of free and open source software development," *First Monday*, vol. 10, no. 2, February 2005.

[4] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of Open Source software development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 3, pp. 309–346, 2002.

[5] T. T. Dinh-Trong and J. M. Bieman, "The FreeBSD project: A replication case study of Open Source development," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 481–494, June 2005.

[6] C. Jensen and W. Scacchi, "Modeling recruitment and role migration processes in OSSD projects," in *Proceedings of 6th International Workshop on Software Process Simulation and Modeling*, St. Louis, May 2005.

[7] G. Robles, J. M. González-Barahona, and M. Michlmayr, "Evolution of volunteer participation in libre software projects: evidence from Debian," in *Proceedings of the 1st International Conference on Open Source Systems*, Genoa, Italy, July 2005, pp. 100–107.

[8] G. Robles and J. M. González-Barahona, "Contributor turnover in libre software projects," in *Open Source Systems Conference, June 8-10, 2006, Como, Italy*, 2006, pp. 273–286.

[9] G. Robles, S. Koch, and J. M. González-Barahona, "Remote analysis and measurement of libre software systems by means of the CVSAnalY tool," in *Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS)*, Edinburgh, Scotland, UK, 2004, pp. 51–56.

[10] G. Robles, J. M. González-Barahona, and J.-J. Merelo, "Beyond executable source code: The importance of other source artifacts in software development (a case study)," *Journal of Systems and Software*, vol. 80, no. 9, pp. 1233–1248, September 2006.

[11] R. A. Ghosh and V. V. Prakash, "The orbiten free software survey," *First Monday*, vol. 5, no. 7, May 2000.

[12] S. Koch and G. Schneider, "Effort, cooperation and coordination in an open source software project: GNOME," *Information Systems Journal*, vol. 12, no. 1, pp. 27–42, 2002.

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. Little seems to be known about productivity in large, long-lived Free/Libre/Open Source Software (FLOSS) projects. 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 FLOSS projects with current size ranging between between 0.6 and 5,3 million lines of code (MLOC). The extracted data does not fit well the cost estimation model COCOMO 81 for proprietary software, motivating the need for FLOSS-specific productivity models. As a first approximation, we evaluated 16 linear regression models involving different pairs of attributes. The major finding of this article is that, if one removes large jumps, a linear model of development (e.g., by expressing effort in contributor-months as a linear function of the number of files) can fit most of the FLOSS projects we analysed. The best effort model we found, had a coefficient of determination $R^2 = 0.79$. One of the more complex FLOSS projects we analysed, namely Eclipse, behaved rather differently than the others, suggesting that different sets of models may be needed for different categories of FLOSS projects. As a by-product of this study, for individual FLOSS projects, we were able to identify growth models with a high degree of accuracy ($R^2 \geq 0.98$).

Keywords: baselines, COCOMO, effort estimation, empirical studies, free software, metrics, open source, productivity.

1 Introduction

Software development productivity measurement and cost estimation has been a research topic for more than 3 decades [1] [2] [3]. The vast majority of empirical studies involved data from proprietary software projects [4]. An increasing number of governments, non-governmental organisations and companies seem interested in using, evaluating and contributing to FLOSS. However, FLOSS communities do not seem to study or use, in general, effort estimation models or other measurement-based models [4]. As a research topic, the exploration

and quantification of productivity may help in comparing FLOSS projects with proprietary systems. Such productivity modelling can also help to identify a baseline to measure the possible impact of changes in, for example, processes, methods and tools used by FLOSS communities. It may also help, in some way, in planning the evolution of future FLOSS projects. This article[1] presents our initial exploratory results in the study of the relationship between size, effort, duration and number of contributors in a very small subset of large, long-lived FLOSS projects. It is difficult to measure effort accurately in FLOSS [6] because we do not know the degree of involvement of each contributor (Was it full-time? Was it some level of part-time?). For this reason, the results reported here are preliminary. Additional research is needed to achieve more reliable productivity and estimation models. We hope, however, that our contribution can help establish FLOSS productivity measurement as a line of research and, eventually, lead to practical tools for the FLOSS communities and other interested parties.

2 Selected projects and measurements

Our research question was how much effort, on average, would be required to develop 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 eleven FLOSS projects we considered, together with the programming language(s) primarily used for each system and the life span, that is, the time between the first publicly recorded commit to the code repository (CVS or Subversion) and data extraction. The date of data extraction was October 2008, with exception of one system (Eclipse) for which it was April 2008.

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

[1] This is an extended and revised version of an earlier extended abstract [5].

The measurements extracted for this study are listed in Table 2. We used the SLOCCount tool[2] 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[3], which stores information extracted from the version control log (CVS or Subversion) in a MySQL database. Specifically, we indirectly used the data from CVSAnalY by downloading databases from the FLOSSMetrics project[4].

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
$EFFORT$	effort in contributor-years	CVSAnalY
DUR	time length of 'active' evolution in years	CVSAnalY
DEV	number of distinct contributors	CVSAnalY

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 get person-years, which is the value used for the total effort spent into a system ($EFFORT$). As said, 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). However, in some projects one 'gate keeper' submits code developed by others, with the effect of turning our values into an underestimation. This is 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 IDEs or other tools that record automatically the amount of effort.

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 large *jumps*. This is illustrated for one of the systems (GCL) on the left of Figure 1. It is unlikely that the productivity of the team of contributors has increased so suddenly on a particular month. It could be that code added in the jump was created and evolved in a private space (or as a branch separated from the main version or *trunk*). It could also

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

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

[4] `data.flossmetrics.org`

be that each jump corresponds to external events such as when the repository receives chunks of externally generated code. An alternative hypothesis is that many non-code files were added to the repository, since $FILES$ includes all files, code and non-code.[5] It wouldn't be surprising if the presence of jumps is typical for FLOSS processes, where external code is borrowed and re-used. The implication for our research question is that measuring the total size, including the 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. We also subtracted the initial size value, since we do not have any records of effort for this initial commit. Figure 1 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, where the assumed value of DUR was 11.2 instead or 19.9 years.

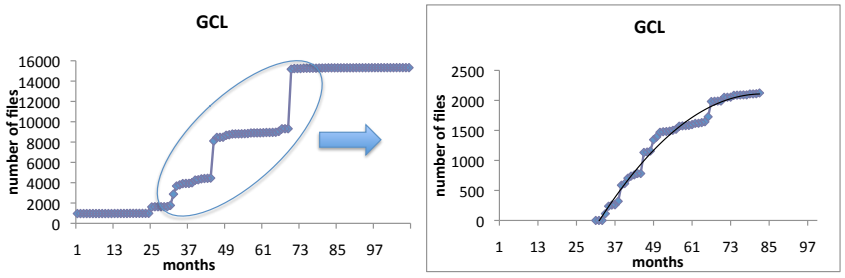


Fig. 1. Trend in FILES (left) and netFILES (right) per month for GCL. The right-hand-side plot corresponds to the central part of the left-hand-side plot, after major 'jumps' have been removed.

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. One contributor-month was counted whenever a developer made at least one commit in a given month. This is a rough approximation, as we explain in section 5.1. We computed two variants, one measuring all the contributors, called DEV[100], and one measuring the number of developers which provided

[5] 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 required effort beyond our availability.

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 a few defect fixes only 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 ranging from 0.98 to 0.99, as can be seen in Table 3.[6] (For GCC and GCL, the 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. This mixture of different trends is generally in agreement with previous empirical studies of FLOSS growth trends [7]. Our finding suggests that 'filtering out' jumps when modelling growth trends might be helpful, provided that the analyst has checked the nature of the jumps. For other purposes, the detection (as opposed to removal) of jumps can provide quite useful information for understanding the evolution of a FLOSS system.

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

The 'filtered' trends can be used as a simple estimation model of the amount of new functionality implemented per month, assuming that there are no rad-

[6] 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, possibly more appropriate or meaningful, measures such as the mean (or median) magnitude of relative error (MMRE) or the Akaike Information Criterion (AIC).

ical changes in the FLOSS project such as, for example, the departure of key members in the core team, or important architectural restructurings. These growth trend models are usable for a single project and can be seen as *specific* estimation models for each FLOSS project.

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.

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 proprietary systems. As a starting point, we used here its earliest and possibly simplest version [2], called COCOMO 81, published in 1981 and calibrated to data from 63 proprietary projects. 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* similar to the size range of the studied FLOSS systems, even though the 63 projects used to calibrate COCOMO are likely to be smaller in size than our FLOSS sample[7]. 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 in FLOSS than in proprietary systems. Only the project with smallest $KLOC$ in the sample (GIMP) is close to COCOMO's prediction. There is evidence in [8] 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 while COCOMO represents software built from scratch. The FLOSS projects do not have, in general, a fixed schedule and budget and they continue to evolve as long as there is interest 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

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

number of people in the core team), only 1 out of 11 projects (namely GIMP) exceeds the team size predicted by COCOMO.
- There are no clear economies or diseconomies of scale (i.e., sub or superlinearity when KLOC is plotted on the x-axis) in the FLOSS data. (The COCOMO model for $EFFORT$ vs $KLOC$ shows a slight diseconomy of scale. The COCOMO models for DUR and DEV suggest economies of scale).
- Overall, the COCOMO model does not seem to be a good fit for the 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 0.340.

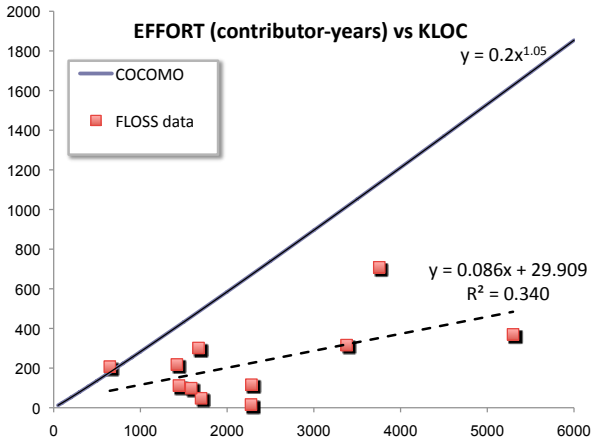


Fig. 2. EFFORT vs KLOC: FLOSS data (N=11) and COCOMO model.

3.3 FLOSS-based estimation models

In search for FLOSS-based generic estimation models we studied the linear correlation between size (measured in $KLOC$ and $FILES$) 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 not meaningful. The best models are obtained when Eclipse is excluded from the dataset, and are indicated in bold.

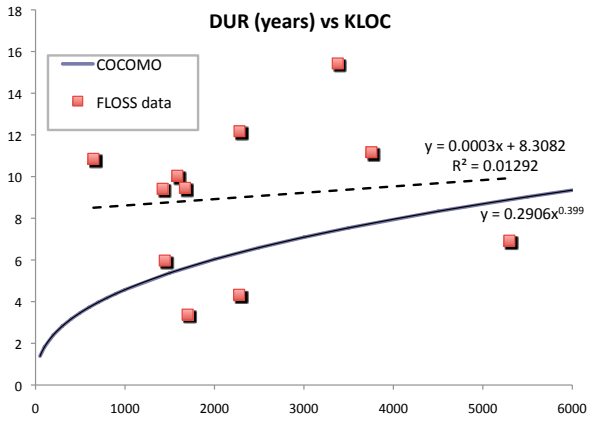


Fig. 3. DUR vs KLOC: FLOSS data (N=11) and COCOMO model.

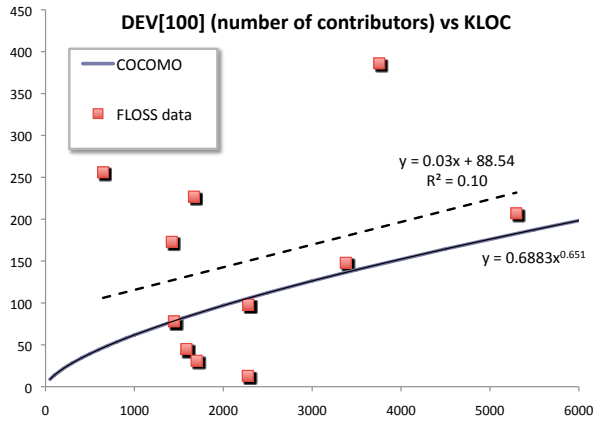


Fig. 4. DEV[100] vs KLOC: FLOSS data (N=11) and COCOMO model.

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

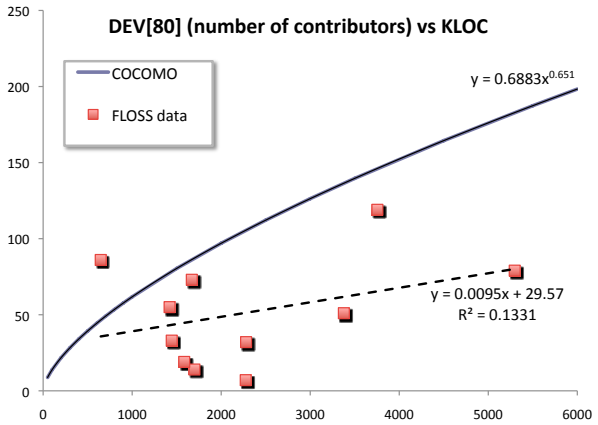


Fig. 5. DEV[80] vs KLOC: FLOSS data (N=11) and COCOMO model.

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, supporting what we observed in Figure 3 that FLOSS projects generate new code at very different rates.

It was the exclusion of the largest system, Eclipse, which appeared to be an outlier in some of the plots, that led to improvement in the goodness of fit in 13 out of 16 linear models considered. All the best regression results correspond to net size and with Eclipse excluded. The best regression model obtained is the one involving $EFFORT$ as a linear function of $netFILES$ (R^2 value of 0.797). Figures 6 and 7 show the data (scatter diagram) and the best linear model for $EFFORT$ vs $netFILES$, considering and excluding Eclipse, respectively. This suggests 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).

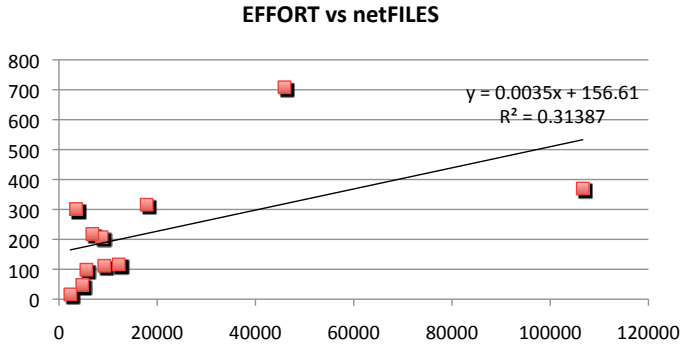


Fig. 6. EFFORT vs netFILES: FLOSS data (N=11) and linear regression model.

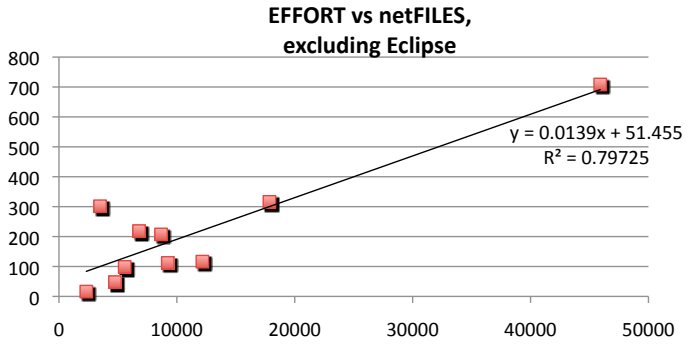


Fig. 7. EFFORT vs netFILES: FLOSS data, excluding Eclipse (N=10) and linear regression model.

3.4 Calculating the 'cost' of 1 MLOC in FLOSS

In order to address our main research question which is how much it takes to develop 1 MLOC in FLOSS, we provide here 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 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. Using this average[8], we can calculate that 1 MLOC will be equivalent to 8,000 files. Taking this value and using the linear 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 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. By finding some good theories, improving the extraction of the data suggested by these, adding more FLOSS systems to the sample and using more advanced techniques than linear regression, it may be possible to achieve more representative values. The simple example illustrated how our very simple FLOSS-based models could be used. We are aware that, since FLOSS projects seek mainly to attract volunteers and not to hire paid professionals, the current practical use of productivity models will be very limited. In the future, however, measurements and models may find their way into helping FLOSS, probably in a yet unforeseen way.

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. [10]). This type of research is frequently called "mining software repositories". To our knowledge, there have been no previous studies of FLOSS software, attempting to quantify the relationships between size, effort, duration and team size. In a position paper, Amor *et al.* [4] 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. Koch [6] studied the impact of tool usage on the efficiency of 30 FLOSS projects from `SourceForge.net`, finding in most of the cases a negative relationship. Our models could be expanded in the future by considering tool usage as a factor that may explain differences in the size-effort and other relationships studied.

The majority of software estimation research has been conducted for proprietary software and initial development [3]. One frequently cited software estimation model, based on proprietary software, is COCOMO [2]. 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. Obviously much more research needs to be done, but this initial finding suggests that FLOSS-specific models are advisable. This

[8] 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 [9].

is not surprising due to the many reported differences between proprietary and FLOSS processes.

The proportion of linear, superlinear and sublinear growth trends (cf. Table 3) is generally in agreement with previous empirical studies of FLOSS growth trends [7]. As far as we know, our work is the first that systematically tries to remove jumps in the monthly growth trends.

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 [11]. 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 committers or developers that 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. We have not quantified how many lines of code have been ported rather than 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 [12], 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 software evolution work (because of the multiplication of work).

5.2 External 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 [13]. 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[9] project database.

5.3 Internal validity

Internal validity threats tend to be present in controlled experiments. Since this has been an observational study reporting on correlations (e.g. between size and number of committers), the severity of this type of threats is likely to be low. We can mention that, as most software, the tools we used in this study to extract and analyse the data may contain defects that may have affected the results.

6 Conclusion and Further work

This paper reports on results of the study of the relationship between size, on the one hand, and effort, duration and team size for large FLOSS, an issue that does not seem to have been empirically studied.

One of the major findings of our study is that, when removing large 'jumps' in the size evolution of FLOSS projects over time, simple regression trends can model this growth with a surprisingly high accuracy. In agreement with previous studies, these trend models reveal either linear, sublinear or superlinear growth (depending on the system studied). FLOSS projects with superlinear growth are particularly interesting from the point of view of cost estimation since they appear to defy the common knowledge that growth rate either remains constant or decreases due to excessive accumulated complexity. Whether different cost models may be suitable for linearly, superlinearly or sublinearly projects remains an open question.

The comparison between data from 11 FLOSS projects and one version of the COCOMO model for proprietary software suggests that generic FLOSS productivity and 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 through refinement and improvement of the measurement approach. We showed how our best models could be used to estimate effort, duration and team size for the development (via evolution) of

[9] `flossmetrics.org`

a 1 MLOC (one million lines of code) system. However, given the volunteer nature of FLOSS we don't expect our models to be used in that way. The best models presented in this paper could provide a baseline to study, for example, the possible impact of new or improved processes, methods and tools. Future models may also help FLOSS communities in systematic planning of the future evolution of their systems and companies planning to contribute to FLOSS, perhaps in some way which is still unforeseen.

In order to improve this research (e.g. increase the external validity), we plan to study an additional number of FLOSS projects. The regression models we used are too simple and do not reflect any theory about FLOSS. Theories based on a deep understanding of the FLOSS world may lead to better models and measuring further attributes that may impact productivity and duration. One of these could be the so-called number of *orphaned lines of code* [8], the portion of the code left behind by contributors who have left a project. Currently, our $FILES$ measurement considers all files in the repository (i.e. 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. proprietary) cost estimation (e.g. [3]) 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).

We plan to examine different approaches to extract the jumps in monthly 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 'jump'. In particular, one needs to check manually the version repositories to confirm whether the 'jumps' are actually corresponding to the inclusion of externally generated code or libraries or the merging with another system.

Acknowledgements

We are grateful to Andrea Capiluppi for comments on an early draft of this paper. We thank the anonymous reviewers for their 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 (2008).

References

1. Wolverton, R.W.: The cost of developing large-scale software. IEEE Trans. Computers **C-23**(6) (June 1974) 615 – 636

2. Boehm, B.: Software Engineering Economics. Prentice Hall (1981)
3. 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, Washington, DC, USA, IEEE Computer Society (2003) 223 –230
4. 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, New York, NY, USA, ACM (2006) 3–6
5. 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, The Netherlands, Eindhoven University of Technology (December, 11-12 2008)
6. Koch, S.: Exploring the Effects of Coordination and Communication Tools on the Efficiency of Open Source Projects using Data Envelopment Analysis. In: Open Source Development, Adoption and Innovation. Volume 234 of IFIP International Federation for Information Processing. Springer Boston (2007) 97 – 108
7. 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), Bari, Italy (March 2006) 206–213
8. 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 (January 2009)
9. Concas, G., Marchesi, M., Pinna, S., Serra, N.: Power-laws in a large object-oriented software system. IEEE Trans. Software Engineering **33**(10) (2007) 687–708
10. Van Rysselberghe, F., Rieger, M., Demeyer, S.: Detecting move operations in versioning information. In: Proc. European Conf. Software Maintenance and Reengineering (CSMR), IEEE Computer Society Press (2006) 271–278
11. Fernandez-Ramil, J., Lozano, A., Wermelinger, M., Capiluppi, A.: Empirical studies of open source evolution. In Mens, T., Demeyer, S., eds.: Software Evolution. Springer-Verlag (2008) 263–288
12. Bellon, S., Koschke, R., Antoniol, G., Krinke, J., Merlo, E.M.: Comparison and evaluation of clone detection tools. IEEE Trans. Software Engineering (September 2007) 577–591
13. 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). (21 April 2006) 6 pp.
14. Lawrence, K.D., Arthur, J.L.: Robust Regression: Analysis and Applications. CRC Press (1990)

Assessing FLOSS Communities. An Experience Report from the QualOSS Project*

Daniel Izquierdo-Cortazar[1], Gregorio Robles[1], Jesus M. Gonzalez-Barahona[1], and Jean-Christophe Deprez[2]

[1] GSyC/LibreSoft, Universidad Rey Juan Carlos, Calle Tulipán s/n, Móstoles, 28933 Madrid, Spain {dizquierdo, grex, jgb}@gsyc.es
[2] Centre of Excellence in Information and Communication Technologies, Rue des Frères Wright 29/3, B-6041 Charleroi - Belgium jean-christophe.deprez@cetic.be

Abstract. This paper presents work done in the QualOSS (Quality of Open Source Software) research project, which aims at building a methodology and tools to help in the assessment of the quality of FLOSS (free, libre, open source) software endeavors. In particular, we introduce the research done to evaluate the communities around FLOSS endeavors. Following the Goal-Question-Metric paradigm, QUALOSS describe goals, the associated questions and then metrics whose measurement helps answer the questions. After applying the QualOSS methodology to four FLOSS projects, namely, Plone, JavaCC, Swallow, and Maemo, some initial conclusions about our approach are drawn.
Keywords: libre software, quality models, product quality, data mining

1 Introduction

QualOSS[2] (Quality of Open Source Software) is a research project focused on the assessment of the quality of FLOSS (free, libre, open source software) endeavor. A FLOSS endeavor is composed of a set of community members (or contributors), a set of work products including code, a set of development processes followed by the community to produce work products, and a set of tools used to support the endeavor, to produce work products and to run the FLOSS software component [9]. In other words, a FLOSS endeavor is really like an enterprise working on

* This work has been funded in part by the European Commission, under the QUALOSS (FP6-IST-5-033547), FLOSSMETRICS (FP6-IST-5-033547) and QUALIPSO (FP6-IST-034763) projects, and by the Spanish CICyT, project SobreSalto (TIN2007-66172).

[2] The QualOSS project is coordinated by CETIC, and includes also University of Namur, Universidad Rey Juan Carlos, Fraunhofer IESE, Zea Partners, UNU-MERIT, AdaCore and PEPITe. The work described in this paper has been performed, or coordinated, mainly by the GSyC/LibreSoft group at Universidad Rey Juan Carlos. More info about the project: http://qualoss.org/

FLOSS development projects. In turn, the exact goal of QualOSS aims at assessing the robustness and evolvability of FLOSS endeavors. One of the key final goals of the project is to allow for the comparison of FLOSS products in a objective, semi-automated, simple and fast way. Robustness and evolvability of FLOSS endeavors are key aspects for software companies that increasingly integrate FLOSS components in their solutions and systems to produce new, reliable software application quickly.

When acquiring software, enterprises are not only interested to know about the product and its quality but also interested in who produced that product and its reputability. For traditional enterprises, reputability can be check based on financial strength of the software provider however, for the FLOSS world, we must find other ways to determine if FLOSS endeavor (or FLOSS project) is serious. This can be done by studying the behavior of a FLOSS community. In particular, a FLOSS community should behaves in a manner to convince potential FLOSS integrators from Industry that it is dependable.

Thus, QualOSS assessment is based on two main concepts, the evaluation of the product, but also the evaluation of the developers and user communities built around them. The first one follows the classic point of view of software metrics studies [11, 16], while the second one considers also the people working and using the product.

Metric	Green	Yellow	Red	Black
NumOfDevelopers	>30	>20	$20 \geq x \geq 3$	<3
NumPosterMailingLists	>60	>40	$40 \geq x \geq 6$	<6
EvolutionNumDev	Fast growing	Growing	Monotonous	Decreasing
TotalNumNonActiveDev	ND	ND	ND	ND
EvolutionNonActiveDev	Monotonous	Weigh growing	Linear growing	Fast growing
TotalNumActiveDev	>30	>20	$20 \geq x \geq 3$	<3
EvolutionNumActiveDev	Fast growing	Growing	Monotonous	Decreasing
NumOfChangesToSource	>300	>150	$150 \geq x \geq 10$	<10
EvolutionChangesToSource	ND	ND	ND	ND
NumOfMessagesOfDev	>50%	>40%	$40\% \geq x \geq 5\%$	<5%
EvolutionMessagesDev	> Monotonous	Monotonous	Decreasing	Fast decreasing

Table 1. Values of several metrics and their color indicators.

FLOSS endeavors provide large amounts of information which can be analyzed, since they are, usually, publicly available [12, 14]. This is the data that QualOSS considers when building the metrics system to connect to its quality model of goals and questions.

The remainder of this paper is as follows: in the next section, related research and state of the art in FLOSS assessment will be presented. Later, the general methodology based on the Goal - Question - Metric approach [4] is introduced, emphasizing on the community side. The fourth section handles with the tools that have been used to obtain the data and presents the projects that have been selected as case studies.

Finally, taking into account the results section, several conclusions are drawn.

2 Related Research

According to the environment for which they have been designed we could classify quality models in two branches. Historically, most of the models were industry-oriented, focused on the development practices found in software development firms, and usually based in product and process metrics. However, with the rise of FLOSS development, some researchers considered that new approaches were needed for considering the different practices found in FLOSS projects. The QualOSS project can be framed in this second branch.
Regarding industry-oriented quality models, several well-known models exist, most notably those by McCall [19] and Boehm [3]. More recently, the quality model of the Deutsche Gesellschaft fur Qualitt [1], the NASA SATC [15], or the ISO 9126 standard [2] can be cited.
In the area of FLOSS, several models have been proposed as well, such as OpenBRR[3] or QSoS[4]. They consider metrics in several realms relevant to FLOSS development and maintenance, ranging from product to process or community metrics. Their main aim is to produce a final mark (or set of marks) which shows their readiness for industrial use. Their main drawback is that, in their current state, they are not supported by automatic processes, which means that putting them to work is a tedious and time-intensive task. In addition, some of them lack the needed benchmarking to fine-tune the methodologies proposed, and in some cases do not consider some important aspects of FLOSS development or maintenance [8].
QualOSS is aimed to fill this gap, providing a set of tools (most probably integrated into a single software platform) and based on an theoretical framework created from scratch but as further explained in the next section, influenced by the existing methodologies mentioned above as well as interviews with FLOSS integrators.

3 Methodology

For achieving its aims, QualOSS started by applying a GQM methodology, from which relevant goals, questions, and finally metrics and indicators were derived. The computation of metrics measurements and aggregation for answering questions of the QualOSS quality model was partially automated with several tools.

[3] `http://www.openbrr.org`
[4] `http://www.qsos.org`

3.1 Interviews and GQM

The first step of the QualOSS project consisted in gathering the state of the art on the topic [5], not only from a mere research point of view, but also identifying tools and metrics [10] that could help in the process of software assessment. In addition, some companies were interviewed to know about their needs, direct or indirect, regarding the quality of software. Those interviews were held with the goal of identifying the needs from an industrial point of view. Companies were classified in two groups, depending on whether their business model is based on FLOSS or not.

For companies with a business model not based on FLOSS, we could determine that they usually focus mainly on the current state of the product. These companies use FLOSS products when, for instance, a product could be modified to achieve some specific customer requirements. Usually, their FLOSS selection process is "ad hoc" and "in situ", in the sense that they are seldom formalized and often based only on the expertise and experience of their technical staff. For these companies the most important attributes of FLOSS projects to be considered are (certainly from a subjective point of view) maturity and stability.

Companies with a business model specifically based on FLOSS are usually not worried to use products still in their pre-production state and with no stable releases. They highlight the importance of the surrounding community and the support it may provide. Therefore these companies do not hesitate to interact with the community, sharing technical and non-technical goals. The licensing model of the product is important for them, too.

With the information obtained from these interviews and their analysis, the ISO 9126 standard was chosen as a starting point, merging in it the criteria that we found relevant for FLOSS, producing the initial QualOSS quality model [6]. In the process, it was found that many important issues surrounding FLOSS endeavor were neither directly linked to the product nor to the development process but rather to the community around the development project. Therefore, the initial QualOSS quality model is an hybrid containing some elements from the *classic* (industry-oriented) software quality models, some others rooted in the industrial requirements on FLOSS, and some others related to FLOSS communities.

The approach for translating the inputs provided by the interviews into parts of the model was the Goal–Question–Metric methodology. The data source of measurement for community aspects are artifacts usually found in FLOSS projects, in particular, SCM (source code management) repositories (Subversion, CVS, etc.) and mailing lists.

Based on the stated goals, questions had to be formulated. Regarding community assessment, we had two main questions. First was "how can we measure community robustness?". Second: "how can we measure community evolvability?". Both questions were further elaborated into several sub-questions, with the aim of characterizing the object of measurement, that is, FLOSS endeavor.

Metric	JavaCC	Maemo	Plone	Swallow
NumOfDevelopers	10	35	151	21
NumPosterMailingLists	ND	1458	ND	ND
EvolutionNumDev	Increasing	Increasing	Increasing	Increasing slowly
TotalNumNonActiveDev	8	9	103	17
EvolutionNonActiveDev	Increasing	Increasing	Increasing	Increasing quickly
TotalNumActiveDev	2	26	49	4
EvolutionNumActiveDev	Decreasing	Irregular	Increasing slowly	Decreasing
NumOfChangesToSource	1,065	55,340	72,587	9,969
EvolutionChangesToSource	very irregular	Decreasing	Increasing	Decreasing
NumOfMessagesOfDev	ND	ND	ND	ND
EvolutionMessagesDev	ND	ND	ND	ND

Table 2. Metrics obtained from each project

In addition, we have taken into account several points of view in the assessment process. We have worked with the traditional roles of *product manager*, *project manager architect analyst*, *developer*, *tester* and *technical writer*. For this study we have just used the role of *product manager* as the point of view to be used. In this way, this role has in mind a long term vision of the project. In fact, this role is more interested to know about the evolution trend of the robustness and evolvability of a FLOSS endeavor rather than to know about the current state of an FLOSS endeavor. This directly fits with the main idea of QualOSS that consists of measuring robustness and evolvability, where it is more important the trend of a project more than the current state.
From this point of view, and following the GQM approach, we have defined several questions to be answered

Community Size and Regeneration Adequacy:

- Has the evolution of new community members reporting bugs remained stable or grown over the history of the FLOSS endeavor? (at least shown a stable or positive trend overall)
- Has the evolution of new code contributing members remained stable or grown over the history of the FLOSS endeavor? (at least shown a stable or positive trend overall)
- Has the evolution of new members contributing data other than code or bug report remained stable or grown over the history of the FLOSS endeavor? (at least shown a stable or positive trend overall)
- Has the evolution of new core contributing members remained stable or grown over the history of the FLOSS endeavor? (at least shown a stable or positive trend overall) (a core member is one with commit right who perform commits frequently for instance, more than once every three month period)
- What is the evolution of core members who stopped contributing for a significant period?
- Has the evolution of core members who stopped contributing for a significant period been compensated by the joining of new core members around the same time frame?

– What is the average longevity of committers to the FLOSS endeavor

Community Interactivity and Workload Adequacy:

– Is the number of events adequate (to show a lively community)?
– Is the number of code commits adequate (to show a lively committer community)?
– Has the size of community supporting older versions of a FLOSS component remained sufficient? (compared to number of bug reports, the number of report fix vs those open and compared to code size)
– Are they sub-groups in the community? If so, are they disconnected or are active community members serving as bridges between these sub groups (sub groups can be at the level of roles such bug reporter, committer but subgroups can also be studied at the code level)

Community Composition Adequacy

– What companies have contributed bug reports?
– What companies have contributed code (code patches or code commits)?
– What companies have employees in leadership position in the FLOSS community (project leader, release manager, treasurer, ...)?
– What companies provide services on the FLOSS component?
– What are the roles who are filled by active community members?
– Are these roles adequate for the FLOSS endeavor in question?
– Is there a sufficient number of code expert on the various portion of the FLOSS component?

3.2 Metrics and indicators

Our manual and tooled methods for gathering information interact with FLOSS data repositories as humans would. That is, we have not asked special access to repositories backends. In turn, this explains why certain metrics have been classified as basic and other as advanced. If we had negotiated direct access to database servers, some metrics currently classified advanced could easily be measured in a matter of seconds. However, the applicability of the QUALOSS methodology would suffer as it would be narrowed to only those FLOSS endeavors that grant direct access to their data. Consequently, we favor the approach where no special access rights are granted for applying the QUALOSS assessment methodology.

For convenience, metrics have been divided into two groups: basic (those that can be directly obtained from a software analysis tool) and advanced (those that cannot, or are too complex). As a rule of thumb, those metrics requiring more than 10 minutes to be measured were defined as advanced. The most significant basic metrics are[5]: number of developers, number of posters to mailing lists, evolution of number of developers, total number of non active developers, evolution of non active developers, total number of active developers, evolution of active

[5] The reader interested in the complete list of metrics, basic and advanced, should refer to [6], where they are listed and explained in detail.

developers, number of changes to source, evolution of number of changes to source, and number of messages of developers.
The most significant advanced metrics are: number of people writing in forums, number of people participating on the IRC, type of license used by the project source code, type of license used by the project documentation, developer regeneration, maturity of the community, messages about security bug issues, developers' forum activity, number of posts about security issues, time lapse between a security bug is reported and the final commit, fraction of commits related to security problems, number of messages in the mailing list about bugs and number of posts about bugs.
Since the amount of metrics (basic and advanced) is high, indicators were defined. An indicator is a color signaling how good a project scores in a given metric. To obtain good values for indicators, experts were interviewed, so at this time they are not based on statistical analysis. In descendant order, from "good" to "bad", indicator colors are green, yellow, red and black. Table 1 provides the threshold values for some metrics and their corresponding indicators. However it should be noticed, that indicators are not universal truth, but simply represent how enterprises perceive risks. Furthermore, an enterprise may tailor indicators to answer a question if they believe the risks in a different way.

3.3 Tools

Several data retrieval and analysis tools are used to calculate the metrics and indicators described above.

CVSAnalY CVSAnalY [21] analyzes source code management (SCM) repositories. Currently it can work with three kinds of repositories: CVS, Subversion and GIT. For each of them, CVSAnalY creates a database that consists of several tables, being the most important:

- **log**: log file from Subversion, CVS or GIT.
- **committers**: committers and related information.
- **files**: files and related information.

MailingListStats MailingListStats analyzes information in a *mbox*, usually obtained from an archive of the e-mail messages sent to a mailing list. It downloads the archives found in a given URL, and extract the relevant information (in *mbox* format) into an SQL database.
The most important tables of MailingListStats are:

- **messages**: messages with all their corresponding metadata (submitter, date, subject, receiver, etc.)
- **mailing_lists**: analyzed mailing lists
- **people**: identified posters

4 Case Studies

The QualOSS quality model has been applied to a set of projects selected by the QualOSS partners, obtaining their relevant metrics and indicators. Since QualOSS is an iterative project itself, with these first evaluations the main aim has been to check the theoretical framework built in the initial approach used in QualOSS. Therefore, the process is still not completely automated. Moreover, the difficulties encountered were noted and will be provided as feedback to the next iteration of the QualOSS project to improve its quality model or how the quality model criteria are evaluated.

Projects were selected for our case studies had to satisfy several selection criteria, of which the following can be highlighted:

- They should cover all the programming languages addressed by QualOSS, namely, Ada, C, C++, Java and Python.
- At least one of them should have been implemented in several programming languages
- They should include projects of several maturity levels. For instance, some should be in their early stages of development.
- They should have enough data to obtain the various basic metrics.

Finally, four projects were selected for this initial assessment: Plone (a cross-platform content management system developed in Python), JavaCC (a parser generator developed in Java that converts a grammar specification to a Java program), Swallow (an abstract layer for peer-to-peer application that helps developers avoiding complex low-level details), and the Hildon Application Framework of Maemo (a platform for use in mobile devices).

5 Preliminary Results and Discussion

The results of applying the methodology described previously to the four selected projects are shown in the following two tables: the values of the metrics (table 2) and their correspondent indicators (table 3).

After all the work done to define the first and second version of the QualOSS quality model, and experience gained from applying it in a semi-manual way [7], some lessons were learned.

It was realized that some of the identified metrics were not informative enough. For instance, TotalNumNonActiveDev (total number of non active developers) was not accepted as a metric, since it is difficult to measure and does not provide useful information. Even for some intuitive definition, this metric proved not to be a good indicator of the current state of the project: the older a project is, the more non active developers will be found. However it makes sense to track the evolution of non active developers to see if their number increases too quickly at certain moment of a FLOSS project.

Some of the metrics were not precise enough. Therefore, some of the definitions were changed, and even some of the goals were slightly modified.

Metric	JavaCC	Maemo	Plone	Swallow
NumOfDevelopers	Red	Green	Green	Yellow
NumPosterMailingLists	ND	Green	ND	ND
EvolutionNumDev	Yellow	Yellow	Yellow	Yellow
TotalNumNonActiveDev	ND	ND	ND	ND
EvolutionNonActiveDev	Yellow	Yellow	Yellow	Red
TotalNumActiveDev	Black	Yellow	Green	Red
EvolutionNumActiveDev	Black	Yellow	Yellow	Black
NumOfChangesToSource	Green	Green	Green	Green
EvolutionChangesToSource	ND	ND	ND	ND
NumOfMessagesOfDev	ND	ND	ND	ND
EvolutionMessagesDev	ND	ND	ND	ND

Table 3. Indicators for the selected projects.

The number of basic metrics were not high enough. Currently, they are useful to provide a first glimpse about a project, but most informative metrics (the advanced set) were not yet available for our initial study. Since there is a direct relationship between tools and metrics, in most cases, the lack of metrics measurements can be tracked to the unavailability of tools good enough to measure them.

An important lesson learned is that the analysis of FLOSS projects can be too wide, and it is difficult to obtain metrics for all the quality attributes that observers could find interesting. Therefore, the decision of focusing only in SCMs and mailing lists (and not considering for example forums, wikis or release files) has proved to help focusing and making the model usable. It is quite important to focus on a few relevant repositories, and try to obtain as many basic metrics that provide information from different point of views.

The naive approach we used for indicators has resulted to be insufficient. It is necessary to have a statistical base to obtain some well calibrated indicators. Indicators actually show the current state of a FLOSS project, so it would be more useful to define such indicators based on empirical studies on many FLOSS projects rather than solely on the opinion of experts. This would increase the consistency of the indicators. The FLOSSMetrics[6] research project is expected to provide a large scale database with metrics from thousands of projects, which could thus be used for this purpose. Some common aspects have been identified between QualOSS and FLOSSMetrics. In particular, using information from both projects, it is expected that indicators can be obtained based on relevant statistical analyses.

As a first step of the further work, a new set of indicators is been designed in the QualOSS project. They directly try, by strong reviews from each partner, to define a set of basic metrics will allow to answer more consistently the set of questions. However not all the metrics will be automatically obtained, this is because we have proposed a lightweight methodology that will commonly be used to compare among several

[6] `http://flossmetrics.org/`

FLOSS alternatives. The associated tool will assist in retrieving the data and computing all the basic metrics, which will in turn be aggregated using our indicators. The second level is provided for those with enough resources to spend more time and devote more effort to some advanced aspects, usually not automatable. It is expected that this heavyweight level will usually be used by FLOSS endeavor themselves to perform an introspection on a particular aspects such as verification and validation or professionalism and eventually identify how these aspects can be improved in the future.
Finally, we must study advanced aspects of community behaviors and how they can be integrated in the QualOSS methodology. For instance, community-driven projects show interesting interactions among participants [20, 13, 18]. Even, more complex analysis regarding the network structure [17] of communities could also be the base for providing additional objective metric related to the community.

References

1. Deutsche Gesellschaft fuer Qualitaet, Software-Qualitaetssicherung. Technical report, VDE-Verlag Berlin, 1986.
2. *ISO/IEC 9126 International Standard, Software engineering-Product quality, Part 1: Quality model*. 2001.
3. E. al. Barry W. Boehm. *Characteristics of Software Quality (TRW series of software technology)*. Elsevier.
4. V. R. Basili. Software modeling and measurement: the goal/question/metric paradigm. Technical report, College Park, MD, USA, 1992.
5. M. Ciolkowski, M. Soto, and J.-C. Deprez. Measurement requirements specifications. specification of goals for the qualoss quality model. Technical report, QualOSS Consortium, 2007.
6. M. Ciolkowski, M. Soto, and J.-C. Deprez. Metrics system and prototype qualoss models. Technical report, QualOSS Consortium, 2007.
7. J.-C. Deprez. Reference f/oss project report. Technical report, QualOSS Consortium, 2007.
8. J.-C. Deprez and S. Alexandre. Comparing assessment methodologies for free/open source software: OpenBRR & QSOS (to appear). In *Proceedings of the 9th International Conference on Product Focused Software Process Improvement (PROFES 2008)*, June 2008.
9. J.-C. Deprez, F. Fleurial-Monfils, M. Ciolkowski, and M. Soto. Defining software evolvability from a free/open-source software. In *Proceedings of the Third International IEEE Workshop on Software Evolvability*, pages 29–35. IEEE Press, October 2007.
10. J.-C. Deprez, J. Ruiz, and I. Herraiz. Evaluation report on existing tools and existing f/oss repositories. Technical report, QualOSS Consortium, 2007.

11. N. Fenton. *Software Metrics: A Rigorous Approach*. Chapman and Hall, 1991.
12. D. M. Germán and A. Mockus. Automating the measurement of open source projects. In *Proceedings of the 3rd Workshop on Open Source Software Engineering*, Portland, Oregon, USA, 2003.
13. J. M. González-Barahona, L. López-Fernández, and G. Robles. Community structure of modules in the apache project. In *Proceedings of the 4th Workshop on Open Source Software Engineering*, Edinburg, Scotland, UK, 2004.
14. J. M. González-Barahona and G. Robles. Free software engineering: A field to explore. *Upgrade Magazine*, IV(4):49–54, Aug. 2003.
15. L. Hyatt and L. Rosenberg. A software quality model and metrics for risk assessment. In *ESA'96 Product Assurance Symposium and Software Product ASS*.
16. S. H. Kan. *Metrics and Models in Software Quality Engineering (2nd Edition)*. Addison-Wesley Professional, September 2002.
17. L. López, G. Robles, J. M. G. Barahona, and I. Herraiz. Applying social network analysis techniques to community-driven libre software projects. *International Journal of Information Technology and Web Engineering*, 1(3):27–48, July-September 2006.
18. G. Madey, V. Freeh, and R. Tynan. Modeling the Free/Open Source software community: A quantitative investigation. In S. Koch, editor, *Free/Open Source Software Development*, pages 203–221. Idea Group Publishing, Hershey, Pennsylvania, USA, 2004.
19. J. A. McCall, P. K. Richards, and G. F. Walters. Factors in software quality. Technical report, 1977.
20. A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of Open Source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, 2002.
21. G. Robles, S. Koch, and J. M. González-Barahona. Remote analysis and measurement of libre software systems by means of the CVSAnalY tool. In *Proc 2nd Workshop on Remote Analysis and Measurement of Software Systems*, pages 51–56, Edinburg, UK, 2004.

FLOSS Communities: Analysing Evolvability and Trustworthiness from an Industrial Perspective

Daniel Izquierdo-Cortazar*, Jesus M. Gonzalez-Barahona*, Gregorio Robles*,
Jean-Christophe Deprez† and Vincent Auvray‡
*GSyC/LibreSoft, Universidad Rey Juan Carlos, Mostoles, Madrid
{dizquierdo,jgb,grex}@libresoft.es
†Centre of Excellence in Information and Communication Technologies, Charleroi, Belgium
jean-christophe.deprez@cetic.be
‡PEPITe, Liège, Belgium
v.auvray@pepite.be

***Abstract*—Plenty of companies try to access Free/Libre/Open Source software (FLOSS) products, but they find a lack of documentation and responsiveness from the libre software community. But not all of the communities have the same capacity to answer questions. Even more, most of these communities are driven by volunteers which in most of the cases work on their spare time. Thus, how active and reliable is a community and how can we measure their risks in terms of quality of the community is a main issue to be resolved. Trying to determine how a community runs and look for their weaknesses is a way to improve themselves and, also, a way to obtain trustworthiness from an enterprise point of view focused on maintenance or evolvability. This paper presents work done in the QualOSS project, which aims at building a methodology and tools to evaluate the communities around FLOSS products (among other quality attributes). Following the Goal-Question-Metric approach, QualOSS describes goals, the associated questions and then metrics whose measurements helps answer those questions. In order to have a statistical basement, around 1400 FLOSS projects have been studied to create thresholds which will help to determine a project's current status compared with this initial set of FLOSS communities.**

***Keywords*-libre software, quality models, product quality, data mining, communities**

I. Introduction

QualOSS [1] (Quality of Open Source Software) is a research project focused on the assessment of the quality of FLOSS (free, libre, open source software) endeavor. A FLOSS endeavor is composed of a set of community members (or contributors), a set of work products including code, a set of development processes followed by the community to produce work products, and a set of tools used to support the endeavor, to produce work products and to run the FLOSS software component [1]. In other words, a FLOSS endeavor is really like an enterprise working on FLOSS development projects. In turn, the exact goal of QualOSS aims at assessing the robustness and evolvability of FLOSS endeavors. One of the key final goals of the project is to allow for the comparison of FLOSS products in a objective, semi-automated, simple and fast way. Robustness and evolvability of FLOSS endeavors are key aspects for software companies that increasingly integrate FLOSS components in their solutions and systems to produce new, reliable software application quickly.

When acquiring software, enterprises are not only interested to know about the product and its quality but also interested in who produced that product and its reputability. For traditional enterprises, reputability can be check based on financial strength of the software provider however, for the FLOSS world, we must find other ways to determine if FLOSS endeavor (or FLOSS project) is serious. This can be done by studying the behavior of a FLOSS community. In particular, a FLOSS community should behaves in a manner to convince potential FLOSS integrators from Industry that it is dependable.

Most research define maintainability as a characteristic of a product. This is acceptable for certain quality characteristic such as reliability and efficiency which are real intrinsic quality characteristic of a product. Although a same approach can apply to maintainability, it often is quite restrictive. Maintainability is a characteristic that is associated to a product and the set of people involved in its implementation. This situation is further emphasized in the FLOSS world. Indeed software implementation is such a human intensive activity that the community of contributors is probably a factor as influential, if not even more influential, than the product went assessing maintainability in the FLOSS world.

Thus, QualOSS assessment is based on two main concepts, the evaluation of the product, but also the evaluation of the developers and user communities built around them. The first one follows the classic point of view of software metrics studies [2], [3], while the second one considers also the people working and using the product.

[1]The QualOSS project is coordinated by CETIC, and includes also University of Namur, Universidad Rey Juan Carlos, Fraunhofer IESE, Zea Partners, UNU-MERIT, AdaCore and PEPITe. The work described in this paper has been performed, or coordinated, mainly by the GSyC/LibreSoft group at Universidad Rey Juan Carlos. More info about the project: http://qualoss.org/

FLOSS endeavors provide large amounts of information which can be analyzed, since they are, usually, publicly available [4], [5]. This is the data that QualOSS considers when building the metrics system to connect to its quality model of goals and questions.

The remainder of this paper is as follows: in the next section, related research and state of the art in FLOSS assessment will be presented. Later, the general methodology based on the Goal - Question - Metric approach [6] is introduced, emphasizing on the community side. This section also deals with the definition of thresholds based on the analysis of hundreds of projects. The fourth section handles with the tools that have been used to obtain the data and presents some results from the dataset selected as case studies in QualOSS. Finally, taking into account the results section, several conclusions are drawn.

II. Related Research

According to the environment for which they have been designed we could classify quality models in two branches. Historically, most of the models were industry-oriented, focused on the development practices found in software development firms, and usually based in product and process metrics. However, with the rise of FLOSS development, some researchers considered that new approaches were needed for considering the different practices found in FLOSS projects. The QualOSS project can be framed in this second branch.

Regarding industry-oriented quality models, several well-known models exist, most notably those by McCall [7] and Boehm [8]. More recently, the quality model of the Deutsche Gesellschaft fur Qualitt [9], the NASA SATC [10], or the ISO 9126 standard [11] can be cited.

In the area of FLOSS, several models have been proposed as well, such as OpenBRR[2] or QSoS[3]. They consider metrics in several realms relevant to FLOSS development and maintenance, ranging from product to process or community metrics. Their main aim is to produce a final mark (or set of marks) which shows their readiness for industrial use. Their main drawback is that, in their current state, they are not supported by automatic processes, which means that putting them to work is a tedious and time-intensive task. In addition, some of them lack the needed benchmarking to fine-tune the methodologies proposed, and in some cases do not consider some important aspects of FLOSS development or maintenance [12]. Finally it is necessary to show that in the case of OpenBRR the number of metrics associated to the community side are just two metrics [13]. On the other hand, QSoS provides four metrics related to activity over the source code [14]

With respect to the risk management and maintenance process, this is a matter of research for a long time [15] where some of the risks affecting software development are related to human resources. In DeMarco and Lister's classic *Peopleware* [16], few chapters are focused on the management of such issues.

Focusing on the academic side, recent research has been carried out on this issue in the case of FLOSS endeavors. Mockus *et al*. identified that in FLOSS projects a small set of people carry on the majority of the software development (they are named as the core group [17]). Some other authors studied the composition of the core group in several FLOSS projects. They found that just few projects show a stable core group, what means that there exist a turnover of developers [18], this is unavoidable and there is a knowledge gap left by the old core group [19]. Thus, in each regeneration of developers, there are a small set of developers who lead the contributions, but for a limited amount of time [20].

QualOSS is aimed to fill this gap [21], [22], and specifically in terms of community assess to provide a methodology whose metrics and indicators are automatically retrieved and all based on a theoretical framework. It will be further explained in the next section. Thus, results are influenced by the existing methodologies and their lack of information regarding communities, what we think that it is a key factor to take into account in software maintenance process, as well as interviews with FLOSS integrators.

III. Methodology

For achieving its aims, QualOSS started by applying a GQM methodology, from which relevant goals, questions, and finally metrics and indicators were derived. The computation of metrics measurements and aggregation for answering questions of the QualOSS quality model was partially automated with several tools.

A. Interviews and GQM

The first step of the QualOSS methodology consisted of gathering the current state of the art on the topic, from a theoretical and empirical point of view [23]. In addition, some companies were interviewed to know about their needs, direct or indirect, regarding the quality of software. Those interviews were held with the goal of identifying the needs from an industrial point of view.

Focusing on the community side, companies with a business model specifically based on FLOSS are usually not directly worried to use products still in their pre- production state and with no stable releases. They highlight the importance of the surrounding community and the support it may provide. Therefore these companies do not hesitate to interact with the community, sharing technical and non-technical goals. The licensing model of the product is important for them, too [24].

With the information obtained from these interviews and their analysis, the ISO 9126 standard was chosen as a starting point, merging in it the criteria that we found

[2]http://www.openbrr.org

[3]http://www.qsos.org

relevant for FLOSS, producing the initial QualOSS quality model [25]. In the process, it was found that many important issues surrounding FLOSS endeavor were neither directly linked to the product nor to the development process but rather to the community around the development project. Therefore, the initial QualOSS quality model is an hybrid containing some elements from the *classic* (industry-oriented) software quality models, some others rooted in the industrial requirements on FLOSS, and some others related to FLOSS communities.

The approach for translating the inputs provided by the interviews into parts of the model was the Goal–Question–Metric methodology. The data source of measurement for community aspects are artifacts usually found in FLOSS projects, in particular we have focused this set of metrics in the SCM (Source Code Management System).

Based on the stated goals, questions had to be formulated. Regarding community assessment, we had two main questions. First was "how can we measure community robustness?". Second: "how can we measure community evolvability?". Both questions were further elaborated into several sub-questions, with the aim of characterizing the object of measurement, that is, FLOSS endeavor.

In addition, we have taken into account several points of view in the assessment process. For this study we have used the role of *product manager* as the point of view to be used. In this way, this role has in mind a long term vision of the project. In fact, this role is more interested to know about the evolution trend of a FLOSS endeavor rather than to know about the current state of an FLOSS endeavor.

From this point of view, and following the GQM approach, we have defined several questions to be answered. Since this is a paper and there is limited space, just some questions will be written down. More information and specific questions can be found at [26]

Community Size and Regeneration Adequacy:

- Has the evolution of new core contributing members remained stable or grown over the history of the FLOSS endeavor? (at least shown a stable or positive trend overall) (a core member is one with commit right who perform commits frequently for instance, more than once every three month period)
- Has the evolution of core members who stopped contributing for a significant period been compensated by the joining of new core members around the same time frame?
- What is the average longevity of committers to the FLOSS endeavor

Community Interactivity and Workload Adequacy:

- Is the number of code commits adequate (to show a lively committer community)?
- Are they sub-groups in the community? If so, are they disconnected or are active community members serving as bridges between these sub groups (sub groups can be at the level of roles such bug reporter, committer but subgroups can also be studied at the code level)

B. Metrics

To answer the set of questions and describe the corresponding community aspects of FLOSS endeavors, the following metrics have been defined[4]:

- **sra7** average number of months where each committer committed,
- **iwa4** proportion of files maintained by a single committer,
- **iwa5** ratio of the number of lines of source code by the number of committers active in the last year,
- **iwa7** proportion of code files committed to in the last year.

Metrics describing the evolution of projects are also created. Dividing the life of a project into intervals, we measure various low level metrics for each time interval. For each project, we then capture the evolution of each low-level metric by computing the slope of the least square line fitting the resulting sequence. The following slope metrics are defined with the corresponding low-level metrics.

- **sra2** number of new code committers in the interval,
- **sra3** number of new non code committers in the interval,
- **sra9** number of active code committers in the interval,
- **iwa1** ratio of the number of commits by the number of committers in the interval,
- **iwa2** ratio of the number of code commits by the number of code committers in the interval.

Additionaly, as the notion of *core committers* was introduced, for each project, we then also compute the slope of the sequence of following low-level metrics:

- **sra4** number of new core committers in the year
- **sra5** number of core committers that quit in the year
- **sra6** difference between sra4 and sra5.

C. Indicators

For each metric m of a project, let us define a high-level risk indicator $r(m)$. This indicator should measure one aspect of the risk taken by a company engaging in a full floss collaboration with the project. The QualOSS project has defined the following four color-coded risk levels, in order of decreasing risk: black, red, yellow and green.

Defining meaningful indicators is not a trivial problem. Before we describe our methodology, let us stress that indicators should not be trusted blindly. In particular, they should not be considered separately, but rather jointly to form an overall risk picture associated to a project.

[4]Each metric's id is created by the quality attribute name and its position: Community **S**ize and **R**egeneration **A**dequacy (sraX) and Community **I**nteractivity and **W**orkload **A**dequacy (iwaX)

Intuitively, an indicator can be classified depending on whether it is:

- an increasing function of the metric, e.g. iwa4 and sra5, or
- a decreasing function of the metric, e.g. sra7, sra2, sra3, sra9, iwa1, iwa2, sra4 and sra6, or
- neither, e.g iwa5 and iwa7.

The classification of each indicator may be seen as arbitrary. For example, one may agree that the risk associated to sra2 generally decreases with increasing value of sra2, but that this no longer holds for very high sra2 as having a large number of new committers may destabilize the code base. However, we feel that the choices made above are reasonable overall. Once an indicator is assumed to be increasing or decreasing, then our indicator definition problem reduces to selecting a partition of the possible metric values into four intervals and associate each to a risk level.

In this paper, we adopt a data-driven approach to defining indicators. Using the notion of quantile, we search for a partition of a metric's values into a given number of intervals with equal probability. If M is a random variable with invertible cumulative distribution function (CDF) $F(m) = P(M \leq m)$, the *pth quantile* is defined by $F^{-1}(p)$. Hence, if m is a metric with invertible CDF, an increasing indicator $r(m)$ is defined by

- black if $m \in \left]F^{-1}(0.75), +\infty\right[$,
- red if $m \in \left]F^{-1}(0.5), F^{-1}(0.75)\right]$,
- yellow if $m \in \left]F^{-1}(0.25), F^{-1}(0.5)\right]$,
- green if $m \in \left]-\infty, F^{-1}(0.25)\right]$.

At each interval boundary, we have to assign the boundary value to one of the intervals. As long as the CDF is well-behaved, this will not affect the risk distribution. Following this rational a decreasing indicator is defined in the same way as an increasing indicator.

Such indicators define relative, and not absolute, risks. A low risk does not imply that the metric value is intrinsically good, just good compared to other projects, e.g. a green value for a decreasing indicator should be interpreted as a statement that the corresponding metric is in the top quartile.

For a metric m with invertible CDF and an indicator r that is neither increasing nor decreasing in m, we partition the range of values of the metric into 8 intervals of equal probability using the $i/8$th quantiles, $i = 2, \ldots, 7$. Then, we define the indicator $r(m)$ as

- black if $m \in \left]-\infty, F^{-1}(0.125)\right] \cup \left]F^{-1}(0.875), +\infty\right[$,
- red if $m \in \left]F^{-1}(0.125), F^{-1}(0.25)\right] \cup \left]F^{-1}(0.75), F^{-1}(0.875)\right]$,
- yellow if $m \in \left]F^{-1}(0.25), F^{-1}(0.375)\right] \cup \left]F^{-1}(0.625), F^{-1}(0.75)\right]$,
- green if $m \in \left]F^{-1}(0.375), F^{-1}(0.625)\right]$.

Again the above indicator defines a relative risk and it should be interpreted as a distance from the median behavior. For instance, a low risk (green) means that the metric is close to its median value, while a high risk (black) means that the metric value is uncommon.

Metric	Unfiltered data	Filtered data
sra7	1422	735
iwa4	1422	735
iwa5	676	289
iwa7	679	291
sra2_{90}	500	289
sra2_{180}	470	285
sra3_{90}	500	289
sra3_{180}	470	285
sra9_{90}	500	289
sra9_{180}	470	285
iwa1_{90}	1152	732
iwa1_{180}	1095	727
iwa2_{90}	500	289
iwa2_{180}	470	285
sra4	1119	729
sra5	1119	729
sra6	1119	729

Table I
NUMBER OF PROJECTS WITH SUFFICIENT DATA IN FLOSSMETRICS TO MEASURE EACH METRIC BEFORE AND AFTER FILTERING

In practice, the CDF of a metric is unknown and we estimate its inverse by

$$\hat{F}^{-1}(p) = m_{\lfloor r \rfloor} + (r - \lfloor r \rfloor)\left(m_{\lceil r \rceil} - m_{\lfloor r \rfloor}\right), \qquad (1)$$

where $m_0, \ldots, m_{N-1}$ is a dataset of metrics ordered by increasing value and $r = p(N-1)$. Note that Equation 1 is defined even if the CDF $F(m)$ is not invertible at p. Hence, we should apply it cautiously, in particular when the metric is discrete.

To compute our metrics and estimate our indicators, we use data collected by the FLOSSMetrics [27][5] project. Many of the open-source projects considered by FLOSSMetrics appear to be very small and are thus not representative of our population of interest. Indeed, we are assessing the risks associated to a full floss collaboration with open-source projects. This precludes very small projects for which, from a business perspective, a fork should be more appropriate. Hence, we choose to filter the FLOSSMetrics data projects with less than four committers, as suggested in [28].

This particular choice of filter appears to increase the quality of our indicators. The effect of filtering on the number of projects with sufficient data in FLOSSMetrics to measure each metric is given in Table I.

Applying our methodology to the filtered FLOSSMetrics dataset, we obtain the indicators given in Tables II, III, and IV. We tested interval sizes of 30, 90 and 180 days (only periods of 90 and 180 days are shown for space reasons) for the slope indicators associated to sra2, sra3, sra9, iwa1 and iwa2. These tables also incorporate some natural constraints on metrics implied by their definition: sra7 ≥ 1, $0 \leq$ iwa4 $\leq$

[5]http://flossmetrics.org

1, iwa5 ≥ 0 and $0 \leq$ iwa7 ≤ 1. Note that slope metrics have no such constraints. Provided the metric CDF is invertible at our points of interest and there is sufficient data, each risk level should have approximately the same number of projects by construction. To assess the validity of each indicator, we thus present the number of projects falling into each risk level in Tables V, VI, and VII. Overall the projects seem to be evenly distributed across the risk levels for all the metrics except sra4, sr5 and sra6. We believe that this phenomenon is explained by the fact that many projects have short duration and the slope of the metrics sra4, sra5 and sra6 is computed based on very few points. As a result, some slope values become very common, leading to non-invertible CDFs.

To illustrate the benefit of filtering, let us consider the construction of the indicator for iwa4 using the original FLOSSMetrics data. Using Equation 1, we obtain $\hat{F}^{-1}(0.25) = 0.693$, $\hat{F}^{-1}(0.5) = 0.93375$ and $\hat{F}^{-1}(0.75) = 1$. Moreover, there are 356 projects in the interval $]-\infty, 0.693]$, 355 in $]0.693, 0.93375]$, 711 in $]0.93375, 1]$ and none in $]1, +\infty]$. In fact, there are 496 projects with iwa4 at 1, representing 34.88% of the projects. Hence, it is not possible to partition the projects into 4 sets with approximately equal size. After filtering, note that only two projects have the value 1.

As stated before, the indicators presented above measure relative risks. This is acceptable for metrics, such as sra7, iwa4, iwa5 and iwa7, that are easy to interpret. For instance, it is intuitively plausible to say that the risk level associated to sra7 is green if the average longevity of a committer is more than a year. For the slope metrics, the situation is less satisfying. The sign of a slope is easily interpretable. However, its magnitude is less so and it is more difficult to justify intuitively the choice of indicators.

Additionally, the length of time interval must be picked and there is little guidance. In Section III-D, we attempt to answer these criticisms by defining alternative metrics and indicators.

D. Beyond slope metrics

Let us analyze slope metrics to gain some insight. Slope metrics are based on low-level metrics that are interpretable. Using least square linear regression, we model a sequence of low-level metric values by a line $y(t) = w_0 + w_1 t$ and use its slope w_1 as our metric. The slope may be written as the difference between the model value at two consecutive time intervals

$$y(t+1) - y(t) = w_1.$$

This suggest to formulate our risk assessment problem directly as a prediction problem. Given project and a low-level metric $m(t)$, we propose to predict the difference between the value of the metric after a certain time interval and the current value. Any algorithm used predict that difference would then define a particular high-level predictive metric. For instance, given a sequence of low-level metric values $m(t), t = 1, \ldots, n$ from a project, we may build a linear model $\hat{m}(t) = w_0 + w_1 t$ and use it to define the predictive metric $\hat{m}(n+1) - \hat{m}(n) = w_1$ (rounded to the closest integer if m is an integer metric). In Section V, we illustrate on selected open-source projects the use of linear models to predict metric differences.

A risk indicator is then naturally defined in terms of differences between consecutive low-level metric values. For example, we may decide that having one new code committer in the next 90 days (see sra2) corresponds to a green risk level. As before, the indicator may be constructed manually or using a data-driven approach. However, unlike the previous case, this new indicator directly depends on the low-level metric and thus inherits its interpretability. Additionally, it does not embed implicit assumptions about the particular model used for the prediction. Indeed, as demonstrated above, using the slope amounts to using a linear model to predict the low-level metric difference. This separation opens up future research directions: advanced models may be used to predict the differences more accurately, while the indicator definition stays the same.

Before starting a full floss collaboration with a project, a company has to pick a time horizon over which risks should be assessed, based on its business objective. This time horizon should determine the choice of time interval for the prediction. For example, we may want to predict the evolution of new code committers after three or six months. If the interval is too short, say one month, the prediction becomes useless from a business perspective. If the interval is too long, it may become impossible to estimate the evolution accurately. In this paper, we consider an interval length for prediction of one year for sra4, sra5 and sra6. For sra2, sra3, sra9, iwa1, and iwa2, we consider 90 and 180 days.

Using the FLOSSMetrics data, let us propose indicators replacing the former slope indicators. For each low-level metric, we pooled the differences between consecutive metric values over all projects with sufficient data in FLOSSMetrics and with more than four committers. The (differences for the) low-level metrics sra2, sra3, sra9, sra4, sra5, and sra6 are integer-valued. Hence, their CDF is not invertible and we may not use quantiles to define a partition of the differences. Histograms of their differences are shown in Figure 1 to 2.

When the interval length is the same, the histograms of sra2, sra3, and sra9 are similar. Keeping the definition of those metrics in mind, we propose to use the same indicator intervals when the interval length is the same. Similarly, we propose to use the same indicator intervals for sra4, sra5, and sra6. For iwa1 and iwa2, we estimate quantiles to search for a uniform risk distribution. These new indicators defined in terms of differences are given in Table VIII, while Table IX gathers counts of the different risk levels observed in the FLOSSMetrics dataset. Although subjective, we believe that

Metric	Indicator			
	Black	Red	Yellow	Green
sra7	$[1, 5.53555]$	$]5.53555, 8.5789]$	$]8.5789, 12.25]$	$]12.25, +\infty]$
iwa4	$]0.87245, +\infty]$	$]0.7124, 0.87245]$	$]0.5522, 0.7124]$	$[0, 0.5522]$
iwa5	$[0, 2130.7] \cup]72295, +\infty]$	$]2130.7, 5293.5] \cup]46029.1, 72295]$	$]5293.5, 9791.7] \cup]29543.5, 46029.1]$	$]9791.7, 29543.5]$
iwa7	$[0, 0.012875] \cup]0.364425, 1]$	$]0.012875, 0.0443] \cup]0.2555, 0.364425]$	$]0.0443, 0.076625] \cup]0.188075, 0.2555]$	$]0.076625, 0.188075]$

Table II
INDICATORS FOR THE METRICS SRA7, IWA4, IWA5, AND IWA7

Metric	Indicator			
	Black	Red	Yellow	Green
$sra2_{90}$	$]-\infty, -0.0833]$	$]-0.0833, -0.0387]$	$]-0.0387, -0.0129]$	$]-0.0129, +\infty[$
$sra2_{180}$	$]-\infty, -0.3167]$	$]-0.3167, -0.1469]$	$]-0.1469, -0.0527]$	$]-0.0527, +\infty[$
$sra3_{90}$	$]-\infty, -0.1993]$	$]-0.1993, -0.0684]$	$]-0.0684, -0.0296]$	$]-0.0296, +\infty[$
$sra3_{180}$	$]-\infty, -0.7]$	$]-0.7, -0.2647]$	$]-0.2647, -0.1135]$	$]-0.1135, +\infty[$
$sra9_{90}$	$]-\infty, -0.0909]$	$]-0.0909, -0.0243]$	$]-0.0243, 0.042]$	$]0.042, +\infty[$
$sra9_{180}$	$]-\infty, -0.2364]$	$]-0.2364, -0.0593]$	$]-0.0593, 0.0901]$	$]0.0901, +\infty[$
$iwa1_{90}$	$]-\infty, -4.0177]$	$]-4.0177, -0.5290]$	$]-0.5290, 1.3411]$	$]1.3411, +\infty[$
$iwa1_{180}$	$]-\infty, -13.4047]$	$]-13.4047, -1.5654]$	$]-1.5654, 4.1337]$	$]4.1337, +\infty[$
$iwa2_{90}$	$]-\infty, -5.5961]$	$]-5.5961, -1.0056]$	$]-1.0056, 0.3397]$	$]0.3397, +\infty[$
$iwa2_{180}$	$]-\infty, -19.4]$	$]-19.4, -3.6696]$	$]-3.6696, 0.8735]$	$]0.8735, +\infty[$

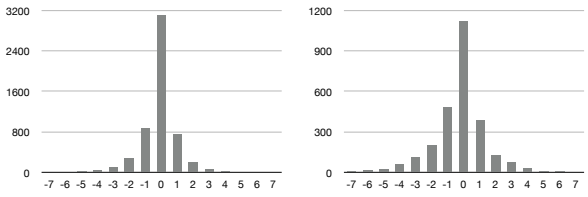
Table III
INDICATORS FOR THE METRICS SRA2, SRA3, SRA9, IWA1, AND IWA2. THE SUBSCRIPT DENOTES THE LENGTH OF THE TIME INTERVAL MEASURED IN DAYS.

Metric	Indicator			
	Black	Red	Yellow	Green
sra4	$]-\infty, -0.5]$	$]-0.5, -0.2]$	$]-0.2, -0.0357]$	$]-0.0357, +\infty[$
sra5	$]0.4, +\infty[$	$]0.1, 0.4]$	$]0, 0.1]$	$]-\infty, 0]$
sra6	$]-\infty, -0.7636]$	$]-0.7636, -0.3]$	$]-0.3, -0.1429]$	$]-0.1429, +\infty[$

Table IV
INDICATORS FOR THE METRICS SRA4, SRA5, AND SRA6

Metric	Indicator				
	Black	Red	Yellow	Green	Total
sra7	184	184	185	182	735
iwa4	184	183	184	184	735
iwa5	73	72	72	72	289
iwa7	74	72	72	73	291

Table V
COUNTS OF THE OBSERVED INDICATOR VALUES FOR THE METRICS SRA7, IWA4, IWA5, AND IWA7 IN FLOSSMETRICS



(a) 90 days (b) 180 days

Figure 1. Histograms of the sra2 differences

our choice of indicators for sra2, sra3, sra9, sra4, sra5, and sra6 is intuitively reasonable. In particular, it does not lead to alarming observed risk distributions. On the other hand, we notice that the risk distribution for the 90 days version of iwa1 and iwa2 appears to be non-uniform. Taking a close look at the data, it appears that the metric CDF is non invertible. In particular, there are 841 samples with a difference of 0 for iwa1 and 574 samples with a difference

	Indicator				
Metric	Black	Red	Yellow	Green	Total
$sra2_{90}$	75	70	72	72	289
$sra2_{180}$	72	71	71	71	285
$sra3_{90}$	73	72	72	72	289
$sra3_{180}$	73	70	71	71	285
$sra9_{90}$	73	72	72	72	289
$sra9_{180}$	72	71	71	71	285
$iwa1_{90}$	183	183	183	183	732
$iwa1_{180}$	182	182	181	182	727
$iwa2_{90}$	73	72	72	72	289
$iwa2_{180}$	72	71	71	71	285

Table VI
COUNTS OF THE OBSERVED INDICATOR VALUES FOR THE METRICS SRA2, SRA3, SRA9, IWA1, AND IWA2 IN FLOSSMETRICS. THE SUBSCRIPT DENOTES THE LENGTH OF THE TIME INTERVAL MEASURED IN DAYS.

	Indicator				
Metric	Black	Red	Yellow	Green	Total
sra4	199	171	177	182	729
sra5	174	172	114	269	729
sra6	183	214	157	175	729

Table VII
COUNTS OF THE OBSERVED INDICATOR VALUES FOR THE METRICS SRA4, SRA5, AND SRA6 IN FLOSSMETRICS

of 0 for iwa2. In both cases, to obtain a risk distribution closer to uniform, we simply suggest to move the value 0 from the red risk level to the yellow risk level. The redefined indicators are given in the rows denoted by $iwa1'_{90}$ and $iwa2'_{90}$ in Table VIII and Table IX.

E. Tools

CVSAnalY [29] analyzes source code management (SCM) repositories. Currently it can work with three kinds of repositories: CVS, Subversion and GIT. For each of them, CVSAnalY creates a database that consists of several tables, being the most important:

- **log**: log file from Subversion, CVS or GIT.
- **committers**: committers and related information.
- **files**: files and related information.

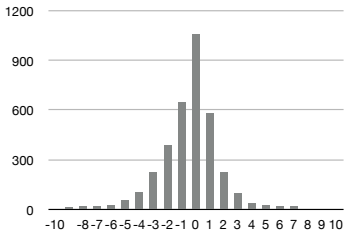


Figure 2. Histogram of the sra6 differences

	Indicator			
Metric	Black	Red	Yellow	Green
$sra2_{90}$	$]-\infty,-2]$	-1	0	$[1,+\infty]$
$sra2_{180}$	$]-\infty,-3]$	$[-2,-1]$	$[0,1]$	$[2,+\infty]$
$sra3_{90}$	$]-\infty,-2]$	-1	0	$[1,+\infty]$
$sra3_{180}$	$]-\infty,-3]$	$[-2,-1]$	$[0,1]$	$[2,+\infty]$
$sra9_{90}$	$]-\infty,-2]$	-1	0	$[1,+\infty]$
$sra9_{180}$	$]-\infty,-3]$	$[-2,-1]$	$[0,1]$	$[2,+\infty]$
$iwa1_{90}$	$]-\infty,-19.4]$	$]-19.4,0]$	$]0,14.5]$	$]14.5,+\infty]$
$iwa1'_{90}$	$]-\infty,-19.4]$	$]-19.4,0[$	$[0,14.5]$	$]14.5,+\infty]$
$iwa1_{180}$	$]-\infty,-39]$	$]-39,-1]$	$]-1,23.7]$	$]23.7,+\infty]$
$iwa2_{90}$	$]-\infty,-20.3]$	$]-20.3,0]$	$]0,13.5]$	$]13.5,+\infty]$
$iwa2'_{90}$	$]-\infty,-20.3]$	$]-20.3,0[$	$[0,13.5]$	$]13.5,+\infty]$
$iwa2_{180}$	$]-\infty,-37.2]$	$]-37.2,-2]$	$]-2,18.5]$	$]18.5,+\infty]$
sra4	$]-\infty,-2]$	-1	0	$[1,+\infty]$
sra5	$[1,+\infty]$	0	-1	$]-\infty,-2]$
sra6	$]-\infty,-2]$	-1	0	$[1,+\infty]$

Table VIII
INDICATORS DEFINED IN TERMS OF METRIC DIFFERENCES

	Indicator				
Metric	Black	Red	Yellow	Green	Total
$sra2_{90}$	521	875	3121	1150	5667
$sra2_{180}$	259	695	1517	287	2758
$sra3_{90}$	992	689	2591	1395	5667
$sra3_{180}$	510	601	1246	401	2758
$sra9_{90}$	709	982	2292	1684	5667
$sra9_{180}$	259	675	1408	416	2758
$iwa1_{90}$	3667	4485	2856	3658	14666
$iwa1'_{90}$	3667	3644	3697	3658	14666
$iwa1_{180}$	1792	1787	1788	1789	7156
$iwa2_{90}$	1417	1894	947	1409	5667
$iwa2'_{90}$	1417	1320	1521	1409	5667
$iwa2_{180}$	690	692	689	687	2758
sra4	610	828	1432	890	3760
sra5	1381	1491	600	288	3760
sra6	960	653	1059	1088	3760

Table IX
COUNTS OF THE OBSERVED VALUES IN FLOSSMETRICS OF THE DIFFERENCE INDICATORS

IV. THREATS TO VALIDITY

As it was said, during the extraction data from FLOSSMetrics database it did not provide full of data regarding other data sources except for the SCM. It means that the results provided for some of the indicators (based on more than one specific metric) are not totally statistical-based. The indicators from mailing lists and BTS were created based on experts opinion and not based on a deep analysis of a set of FLOSS projects. However, what it is presented in this paper is just based on those metrics which are retrieved specifically from the source code management system.

It is also necessary to deal with the fact that projects stored in FLOSSMetrics database do not represent the whole population of FLOSS projects. Most of the projects stored are from Apache, GNOME, KDE and SourceForge projects, which provide a huge set of big, medium and small projects,

but they, again, do not represent the FLOSS world by themselves.

Also, the statistical approach to calculate differences or tendencies in a FLOSS project may not be the best approach, however, the new definition of indicators solves this problem basing these data in just the tendency and not making it dependable from the statistical approach.

Finally, depending on the policy, projects may have a small set of developers who are in charge of committing all the changes. This will skew the results produced by our methodology.

V. Results for Illustration

As an example, table X shows the results for a set of projects directly taken from those which have been used to create indicators. We have focused the results on the periods of 90 and 180 days since these are the longest periods of our study. The main reason for this is that for investment purposes is more interesting to know about how a community will behave during next months, than just during next days. However, the longer the period is, worse accuracy in the results is provided.

As it was said, the indicators are thought as a way to explain the current tendency of the community in terms of robustness and evolvability. So, from an industrial perspective it is more reliable the Evince project than the Nautilus, since it has better indicators ("more greens"). It denotes that there is a problem in Nautilus community and there are some risks that may be studied deeper. Going a step ahead, if the metrics are checked, during last years, the Nautilus projects has decreased its number of committers and also its number of contributions, what means that the maintenance done before is slightly decreasing during last months. It is also important to notice that the core committers (those which do most of the work in a given community) have a greater behavior from that industrial perspective than the Nautilus'. Thus, we may conclude looking at table X that there are more intrinsic risks on the Nautilus community than in the Evince community. It may be useful, for instance, to decide if the Nautilus community needs more effort to go on with their maintenance activity or if your company prefer to invest money on a growing community.

On the other hand, using the linear model's approach, Table XI shows the new set of results. For this approach, there is a new notation based on Δ_X: Δ_X sra2$_Y$ where Δ_X means that we predict the metric difference over the next X days and sra2$_Y$ means that we use a linear model estimated with an interval length of Y days

For example, if the model for iwa1 built with an interval of 30 days is iwa1 = 0.2 - 0.1 t. Then we predict a difference over 30 days of -0.1, over 90 days of -0.3=-0.1*3 and over 180 days of -0.6=-0.1*6. If the metric we want to predict is discrete, then we round the continuous difference prediction to the closest integer.

This approach shows different results from the previous one. In this case, all the communities show a similar behavior in terms of tendency. If we check Table X the differences in the slope for all the projects are not really huge. Thus, it makes sense, since there is an approximation to the closest integer as aforementioned. We believe that this new approach polishes the first one and results are more accurate to the real world. Projects presented for illustration are well known projects and even when the Evince community is more active than the Nautilus's or HTTPD1.3's, all of them show a similar activity in terms of commits per committer, handled files and other metrics.

VI. Conclusions

We have presented a way to estimate "quality" from an industrial perspective based on statistical analysis of hundreds of FLOSS projects. FLOSS communities are key actors in the software evolution and maintenance process and better understanding their behavior through their life will improve the make decision process.

For instance, checking the tendency by means of the methodology explained in this paper, we are able to know if a community is growing, or if the number of core committers is decreasing over and over. Taking into account the latter, we may not be interested in adding more effort in a given project due to the fact that main developers are leaving the project and this is a tendency checked during last months or years. On the other hand, perhaps we are interested in a specific product and we know that some of its weaknesses are motivated because of a really high turnover of developers. It is also important to check the status of potential activity per committers. Sometimes it is necessary, for maintenance purposes to check how many files are being handled by committers in order to check if they are overloaded.

Thus, we can check how reliable are the FLOSS communities looking at the values for the given set of metrics. As it was mentioned, indicators define relative, and not absolute, risks. A low risk does not mean that the metric value is good, this is just good compared to other projects. In this case, this is useful if we are interested in guessing the activity of the community, the general tendency and how it behaves compared to some other set of projects.

VII. Further Work

Indicators must be polished by using new data from FLOSSMetrics (current status of the Melquiades database [6] shows an increase of 600 projects since results were retrieved for this paper). There are other publicly available data sources which may be checked in order to add more accuracy to this analysis. Specifically OSSMole [7] or Ohloh [8] are some

[6] http://melquiades.flossmetrics.org
[7] http://ossmole.sourceforge.net/
[8] http://www.ohloh.net/

Metric	evolution	evince	nautilus	httpd1.3
sra7	8.6044 Y	5.0272 B	7.4484 R	15.7206 G
iwa4	0.3666 G	0.4216 G	0.3168 G	0.0923 G
iwa5	374772.4 B	48282.8 R	246991.9 B	-
iwa7	0.145 G	0.1904 Y	0.0491 Y	6.0e-4 B
$sra2_{90}$	-0.1008 B	0.0346 G	-0.0701 R	-0.0463 R
$sra2_{180}$	-0.3958 B	0.1015 G	-0.3072 R	-0.1839 R
$sra3_{90}$	-0.079 R	0.2457 G	0.0214 G	-0.0474 Y
$sra3_{180}$	-0.2987 R	0.8158 G	-0.0045 G	-0.1891 Y
$sra9_{90}$	0.0257 Y	0.1379 G	-0.0735 R	-0.276 B
$sra9_{180}$	-0.0203 Y	0.3609 G	-0.2445 B	-0.636 R
$iwa1_{90}$	-0.23327969 Y	-0.0312513 Y	-0.20157016 Y	-0.62113349 R
$iwa1_{180}$	-0.61774331 Y	-0.24498722 Y	-0.73020034 Y	-2.19266004 R
$iwa2_{90}$	-1.03448107 R	0.31998312 Y	-1.60891953 R	-0.76343814 Y
$iwa2_{180}$	-2.47361739 Y	-0.15635451 Y	-5.50437436 R	-2.71498719 Y
sra4	0.5 G	1.5636 G	0.1091 G	-0.3582 R
sra5	1.2364 B	1.4848 B	1.5091 B	0.0132 Y
sra6	-0.7364 R	0.0788 G	-1.4 B	-0.3714 R

Table X
ILLUSTRATION IN SOME PROJECTS USING SLOPE APPROACH

Pred. diff.	evolution	evince	nautilus	httpd1.3
Δ_{90} $sra2_{90}$	0 Y	0 Y	0 Y	0 Y
Δ_{180} $sra2_{180}$	0 Y	0 Y	0 Y	0 Y
Δ_{90} $sra3_{90}$	0 Y	0 Y	0 Y	0 Y
Δ_{180} $sra3_{180}$	0 Y	1 Y	0 Y	0 Y
Δ_{90} $sra9_{90}$	0 Y	0 Y	0 Y	0 Y
Δ_{180} $sra9_{180}$	0 Y	0 Y	0 Y	-1 R
Δ_{90} $iwa1_{90}$	-0.23327969 R	-0.0312513 R	-0.20157016 R	-0.62113349 R
Δ_{180} $iwa1_{180}$	-0.61774331 Y	-0.24498722 Y	-0.73020034 Y	-2.19266004 R
Δ_{90} $iwa2_{90}$	-1.03448107 R	0.31998312 Y	-1.60891953 R	-0.76343814 R
Δ_{180} $iwa2_{180}$	-2.47361739 R	-0.15635451 Y	-5.50437436 R	-2.71498719 R
Δ sra4	1 G	2 G	0 Y	0 Y
Δ sra5	1 B	1 B	2 B	0 R
Δ sra6	-1 R	0 Y	-1 R	0 Y

Table XI
ILLUSTRATION IN SOME PROJECTS USING A LINEAR MODEL

examples which have been used for academic purposes. The FLOSSMetrics projects is currently adding data from BTS what means that some other indicators will be added using the statistical approach defined here.

We also need to include advanced aspects of community behaviors and how they can be integrated in the QualOSS methodology. For instance, community-driven projects show interesting interactions among participants [17], [30], [31]. Even, more complex analysis regarding the network structure [32] of communities could also be the base for providing additional objective metric related to the community.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their comments. This work has been funded in part by the European Commission, under the QUALOSS (FP6-IST-5-033547) and FLOSSMETRICS (FP6-IST-5-033547) projects

REFERENCES

[1] J.-C. Deprez, F. Fleurial-Monfils, M. Ciolkowski, and M. Soto, "Defining software evolvability from a free/open-source software," in *Proceedings of the Third International IEEE Workshop on Software Evolvability*. IEEE Press, October 2007, pp. 29–35.

[2] N. Fenton, *Software Metrics: A Rigorous Approach*. Chapman and Hall, 1991.

[3] S. H. Kan, *Metrics and Models in Software Quality Engineering (2nd Edition)*. Addison-Wesley Professional, September 2002.

[4] D. M. Germán and A. Mockus, "Automating the measurement of open source projects," in *Proceedings of the 3rd Workshop on Open Source Software Engineering*, Portland, Oregon, USA, 2003.

[5] J. M. González-Barahona and G. Robles, "Free software engineering: A field to explore," *Upgrade Magazine*, vol. IV, no. 4, pp. 49–54, Aug. 2003.

[6] V. R. Basili, "Software modeling and measurement: the goal/question/metric paradigm," College Park, MD, USA, Tech. Rep., 1992.

[7] J. A. McCall, P. K. Richards, and G. F. Walters, "Factors in software quality," Tech. Rep., 1977.

[8] E. al. Barry W. Boehm, *Characteristics of Software Quality (TRW series of software technology)*. Elsevier.

[9] "Deutsche Gesellschaft fuer Qualitaet, Software-Qualitaetssicherung," VDE-Verlag Berlin, Tech. Rep., 1986.

[10] L. Hyatt and L. Rosenberg, "A software quality model and metrics for risk assessment," in *ESA'96 Product Assurance Symposium and Software Product ASS*.

[11] *ISO/IEC 9126 International Standard, Software engineering-Product quality, Part 1: Quality model*, 2001.

[12] J.-C. Deprez and S. Alexandre, "Comparing assessment methodologies for free/open source software: OpenBRR & QSOS (to appear)," in *Proceedings of the 9th International Conference on Product Focused Software Process Improvement (PROFES 2008)*, June 2008.

[13] O. Consortium, "Business readiness rating for open source," Carnegie Mellon West Center for Open Source Investigation, CodeZoo, SpikeSource and Intel, USA, Tech. Rep., 2005.

[14] A. Origin, "Method for qualification and selection of open source software," Atos Origin, France, Tech. Rep., 2006.

[15] B. W. Boehm, "Software risk management: Principles and practices," *IEEE Softw.*, vol. 8, no. 1, pp. 32–41, 1991.

[16] T. De Marco and T. Lister, *Peopleware : Productive Projects and Teams, 2nd Ed.* Dorset House Publishing Company, Incorporated, 1999.

[17] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of Open Source software development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 3, pp. 309–346, 2002.

[18] G. Robles, "Contributor turnover in libre software projects," in *Proceedings of the Second International Conference on Open Source Systems*, 2006.

[19] D. Izquierdo-Cortazar, G. Robles, F. Ortega, and J. M. Gonzalez-Barahona, "Using software archaeology to measure knowledge loss in software projects due to developer turnover," *Hawaii International Conference on System Sciences*, vol. 0, pp. 1–10, 2009.

[20] M. Michlmayr, G. Robles, and J. M. Gonzalez-Barahona, "Volunteers in large libre software projects: A quantitative analysis over time," in *Emerging Free and Open Source Software Practices*, S. K. Sowe, I. G. Stamelos, and I. Samoladas, Eds. Hershey, Pennsylvania, USA: Idea Group Publishing, 2007, pp. 1–24.

[21] D. Izquierdo-Cortazar, G. Robles, J. M. González-Barahona, and J.-C. Deprez, "Assessing floss communities: An experience report from the qualoss project," in *OSS*, 2009, p. 364.

[22] M. Soto, D. Izquierdo-Cortazar, and M. Ciolkowski, "Measuring the performance of open source development communities: The qualoss approach," in *DASMA Metrik Kongress*, 2009.

[23] J.-C. Deprez, J. Ruiz, and I. Herraiz, "Evaluation report on existing tools and existing f/oss repositories," QualOSS Consortium, Tech. Rep., 2007. [Online]. Available: http://www.qualoss.org/about/Progress/deliverables/WP1_Deliverable1.1.pdf

[24] M. Ciolkowski, M. Soto, and J.-C. Deprez, "Measurement requirements specifications. specification of goals for the qualoss quality model," QualOSS Consortium, Tech. Rep., 2007. [Online]. Available: http://www.qualoss.org/about/Progress/deliverables/WP1_Deliverable1.2_final.pdf

[25] ——, "Metrics system and prototype qualoss models," QualOSS Consortium, Tech. Rep., 2007. [Online]. Available: http://www.qualoss.org/about/Progress/deliverables/WP1_Deliverable1.3.pdf

[26] J.-C. Deprez, "Reference f/oss project report," QualOSS Consortium, Tech. Rep., 2008. [Online]. Available: http://www.qualoss.org/about/Progress/deliverables/WP4_Deliverable4.1_submitted.pdf

[27] I. Herraiz, D. Izquierdo-Cortazar, and F. Rivas-Hernández, "Flossmetrics: Free/libre/open source software metrics," in *CSMR*, 2009, pp. 281–284.

[28] I. Herraiz, "A statistical examination of the evolution and properties of libre software," Ph.D. dissertation, Universidad Rey Juan Carlos, 2008, http://purl.org/net/who/iht/phd.

[29] G. Robles, S. Koch, and J. M. González-Barahona, "Remote analysis and measurement of libre software systems by means of the CVSAnalY tool," in *Proc 2nd Workshop on Remote Analysis and Measurement of Software Systems*, Edinburg, UK, 2004, pp. 51–56.

[30] J. M. González-Barahona, L. López-Fernández, and G. Robles, "Community structure of modules in the apache project," in *Proceedings of the 4th Workshop on Open Source Software Engineering*, Edinburg, Scotland, UK, 2004.

[31] G. Madey, V. Freeh, and R. Tynan, "Modeling the Free/Open Source software community: A quantitative investigation," in *Free/Open Source Software Development*, S. Koch, Ed. Hershey, Pennsylvania, USA: Idea Group Publishing, 2004, pp. 203–221.

[32] L. López, G. Robles, J. M. G. Barahona, and I. Herraiz, "Applying social network analysis techniques to community-driven libre software projects," *International Journal of Information Technology and Web Engineering*, vol. 1, no. 3, pp. 27–48, July-September 2006.

A study of the properties of libre software

Israel Herraiz
Grupo de Sistemas y Comunicaciones
Universidad Rey Juan Carlos, Spain
herraiz@gsyc.urjc.es

Abstract

Software growth (and more broadly, software evolution) is usually considered in terms of size or complexity of source code. However in different studies, usually different metrics are used, which makes it difficult to compare approaches and results. In addition, not all metrics are equally easy to calculate for a given source code, which leads to the question of which one is the easiest to calculate without losing too much information. To address both issues, in this FLOSSMetrics study report I present a comprehensive study, based on the analysis of 52,314 source code files written in C, calculating several size and complexity metrics for all of them. For this sample, I have found double Pareto statistical distributions for all metrics considered (this confirms previous results), and a high correlation between any two of them. This would imply that any model addressing software growth should produce this Pareto distributions, and that analysis based on any of the considered metrics should show a similar pattern, provided the sample of files considered is large enough. This study is based in a work already published in the Mining Software Repositories (MSR) Working Conference in 2007, and in my PhD thesis.

1 Introduction

One of the goals of software engineering is to measure different aspects of software projects, with the aim of finding a small set of attributes that may characterize them. Among those attributes, metrics of the internal attributes of the source code are usually considered, with special attention to size and complexity.

In fact, many different metrics for size and complexity do exist, and have been successfully used in many empirical studies. However, due to this diversity in metrics, comparison of results is not always easy, and the basic question of which metrics are enough to understand a certain aspect of the source code of a project is still largely unsolved. For obtaining some insight in both issues, we I studied a large quantity of source code (about 100,000 files) corresponding to 100 software projects stored in the aggregated database of FLOSSMetrics [8].

All the software included in the study is libre (free, open source) software, which could lead to some bias in the results, but probably they can easily be extrapolated to at least C code of any kind, since the license of the software is not likely to influence distributions of size or complexity. Probably the results can also be extended to other languages different from C, but further research is needed for that conclusion.

FLOSSMetrics is of course not the only data source that may be used for a study like this. There are large collections of libre software in the form of distributions (for instance, several Linux distributions like Debian, Fedora, Ubuntu, Mandriva, etc.). In this study, I have chosen FLOSSMetrics because it is easy to milk the databases to obtain information regarding a large amount of software projects. In the case of this study, that information is source code metrics.

With the quantity of source code measured in FLOSSMetrics (more than 100,00 files, and 32 MSLOC in total, more than 70% of them corresponding to C code), it is possible to apply statistical methods to find out correlations and patterns in the set of analyzed data. In the case of this paper, my first motivation was to find out which independent metrics may be used to characterize size and complexity. I have also confirmed that all metrics considered follow a statistical distribution (double Pareto), which I already found for a set of source code files obtained from FreeBSD [7, 6].

In this respect, it is also worth mentioning that some authors have proposed models to explain why these distributions appear in some of those fields. I have found that those models could be easily applied to the case of software growth, and could be used to simulate the growth of a software product and to model events in a source control management system.

The rest of the paper is as follows. Next section reviews some previous work related to this study. Third and fourth sections describe the data sources and the methodology used. Then, main results and findings are presented,

including a discussion on the suitability for modeling software growth of some models based on the statistical distributions found in our study, which are used in other domains. Finally, some conclusions and directions for further research are offered.

2 Related work

In 2000, Godfrey [5] pointed out that the Linux kernel was growing with a pattern which did not corresponded to Lehman's laws [12]. In a latter work, Lehman [11] qualified the case of Linux as an anomaly. Furthermore, some claimed that the studies were not comparable, because they used different metrics (Godfrey used SLOC, Lehman number of modules, which are actually number of source code files not counting header files).

This case shows how the issue of which metrics are suitable to empirically study a software project is important. Traditionally, SLOC has been used in the empirical studies of libre software projects [15, 10], while LOC, number of files, or some complexity modules are common in other domains.

Some studies have specifically addressed the problem of comparable metrics. For instance, a previous work where the author of this study participated in [9], found that in the context of software evolution the number of source code files and SLOCs were highly correlated. However, that analysis was performed measuring the total quantity of software in the SCM repository of each project every six months, and then correlating all the points obtained, which means that the studied files were not internally independent (we studied different versions of the same files). To avoid this bias, the study presented here has performed measures on a collection of software at a single point in time. The code included in that collection is not composed of different versions of the same files, but correspond to unrelated files, in many cases written by different developers, and can be considered statistically independent. Therefore, if some dependencies are found among the metrics in this case, we can conclude that it is not because of any internal relationship among the files.

After collecting the metrics for the whole collection, I found that all of them showed determined statistical distribution. This finding is not new, and has been noted in other fields. For instance, Mitzenmacher [13] describes some phenomena that develop different statistical distributions. Among them, the distribution of the size of files in a tree of a filesystem is a *double Pareto distribution*. This is the same distribution that I found for the metrics of the set of projects under study here. It has been also found in similar domains, such as in the files available in web and FTP servers [2]. Mitzenmacher has proposed a theoretical model to explain why the distribution of the size of files follows double Pareto [14], which uses concepts quite close to those found in the regular working of a source control system. Therefore, I suggest that it could be easily adapted to simulate and forecast the growth of software.

The problem of obtaining a model for software growth has also been addressed before. For instance, Turski [17, 18] developed a model based on Lehman's laws. Unfortunately, it is not verified by the growth pattern of some libre software projects. Some other models[16, 4, 1] have tried to characterize the process of software development in the libre software world, but they have not been yet checked against the actual history of a meaningful quantity of software projects.

3 Data sources

All the data needed for this study were obtained directly from the FLOSSMetrics databases[1]. In particular, from the aggregated database of source code management (SCM) systems, that at the time of writing this (March 2009), it contains 100 software projects. Nevertheless, the results shown here can be updated easily when new projects are added to that database.

From all the data available in that database, we extracted only source code metrics. The information contained in the database is historical, this is, contains data for different versions of the same files. I retrieved the values of the metrics for the latest version available of every file in the database.

The 100 projects contained 102,471 files, being 52314 files of them written in C. This nearly 50% of the files accounted for more than 70% of the SLOC.

4 Methodology

After all the metrics were extracted, I obtained some descriptive statistics. I calculated the statistical distribution for each one of the metric, taking as sample all the files written in C language. The distributions obtained for each metric were also characterized to find out if they matched any known pattern or distribution.

The correlations between all the metrics were later calculated, to find out if there are some of them which are not providing further information and could therefore be removed from the set. For this, I considered the value of each metric for each one of the files as a point, and linearly correlated every pair of metrics using least squares regression. The results did not show strong correlations between the metrics.

I repeated the same procedure, but this time for the logarithm of the metrics. The correlation coefficients showed strong relationships among the logarithm of the metrics.

[1] See http://melquiades.flossmetrics.org/wiki/doku.php?id=database for a description

Moreover, the statistical distributions appeared to be very close to a normal distribution.

4.1 Selected metrics

FLOSSMetrics contains a rich set of size and complexity metrics, although complexity metrics are only available for C source code. Because of that, in this study I focus in the case of C, using the whole set provided by FLOSSMetrics, which is the following.

To measure size:

- Source Lines of Code (SLOC)
  As defined in [3]:

  > A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing program headers, declarations, and executable and non-executable statements

  I decided to use this metric because it has been traditionally used in the study of the evolution of libre (free / open source) software projects (for instance [5, 15]).

- Lines of Code (LOC)
  This is the number of lines of program text, regardless it is a comment, a blank line, etc.

- Number of blank lines
  This is the number of lines which do not contain any character, besides spaces, tabulators, etc.

- Number of comment lines
  This is the number of lines that are only comments, not containing any code.

- Number of comments
  This is the number of comment blocks. If a comment occupies several lines, but it was only one block, it is considered only one comment.

- Number of C functions
  This is number of functions inside the file.

To measure complexity:

- McCabe's cyclomatic complexity
  McCabe's cyclomatic complexity is the classical graph theory cyclomatic number, indicating the number of regions in a graph. theory, the number of function returns is the number of exit points of follow inside the function.

- Halstead's length, volume, level and mental discriminations

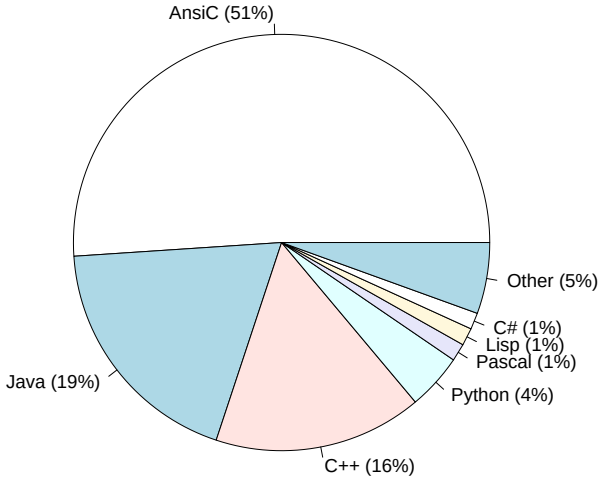


Figure 1. Distribution of programming languages. Values are percentage of source code files in the sample.

A summary of the metrics selected, and the symbols used in the rest of the tables of this paper, is shown in table 1. A thorough definition of these metrics may be found in my PhD thesis [6],

5 Results

The sample contained 102,471 files, distributed by programming language as indicated by figures 1 (for percentages of files) and 2 (for percentages of SLOC). As those figures show, C, C++ and Java account for more than the three quarters of the whole sample.

These 102,471 source code files contained 31,868,184 SLOC. The files were written in 24 different programming languages. Three of them (C, C++ and Java) covered 86% of the files and 90% of the SLOCs.

The average size of the files was 311 SLOC (median: 89 SLOC), with a standard deviation of 1258 SLOC. The largest file contained 340,247 SLOC (340,281 LOC), and the smallest file 0 SLOC (ranging from files with 0 LOC to 1847 LOC).

From the whole sample of files, 52,314 were written in C language.

I found also 73 empty files, that were removed from the sample. Therefore the final sample for all the files written in C language contained 102,398 files.

In the sample I found some files with very high values for some metrics. After inspecting those files, I found many that were not written by humans but automatically generated. Those files are outliers (observations that deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism). This different

Size	Source Lines of Code (SLOC), Lines of Code (LOC), Number of C functions (FUNC), Number of blank lines (BLKL), Number of comment lines (CMTL), Number of comments (CMTN)
Complexity	McCabe's cyclomatic complexity (CYCLO) Halstead's length (HLENG), Halstead's volume (HVOLU) Halstead's level (HLEVE), Halstead's mental discriminations (HMD)

Table 1. Selected metrics for the study

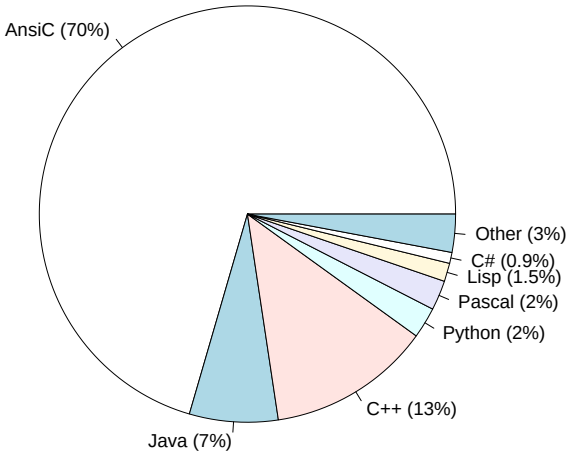


Figure 2. Distribution of programming languages. Values are percentage of SLOC in the sample.

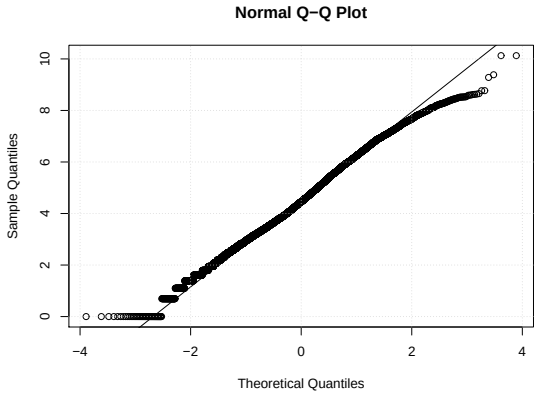


Figure 3. Quantile-Quantile plot for the distribution of the logarithm of SLOC.

mechanism is automatic generation (compared to the usual mechanism of generation by a human developer). As I am interested in the process of software development, I did not consider those files that are not properly part of this process.

Therefore I had to find a method to remove files that are automatically generated. I discarded to do this task by means of heuristics, looking for patterns that are included by some well known code generation tools, such as YACC, a method that would be very time consuming given the amount of files.

I preferred to use statistical methods to remove outliers. The simplest method is the *three sigma rule*. All those files whose values separate more than three times the standard deviation from the mean, are considered outliers. However, this rule can only be applied if the distribution is normal. In this case, the distribution of the metrics were highly right skewed, with a long tail, probably suggesting a power law or lognormal distribution, but not a normal distribution.

I tried firstly to calculate the distributions of the logarithm of the metrics, to find out whether or not the distribution of the metrics was lognormal. To test the normality of the logarithm of the samples, I used the Quantile-Quantile plot. This plot represents the values of the quantiles of our sample against the quantiles of a given reference distribution. If this distribution is the normal distribution, the plot tests the normality of the sample.

After testing all the metrics, all the distributions resulted to be very close to a normal distribution. As an example, I show the test for the distribution of the logarithm of SLOC in figure 3. All the points follow a straight line, although values in the tails seem to deviate from a straight line. In any case, we can consider this distribution to be normal (as the low value of the kurtosis, table 2, evidences as well).

However, this normality test is not robust. It could happen that the distribution is not lognormal but a power law distributions. To find out if the distribution of the logarithm is normal, or if the distribution is a power law, I obtained the *complementary cumulative distribution function* (ccdf). The ccdf of a power law distribution is a straight line, when representing the logarithm of the ccdf against the logarithm of the value (in this case, the considered metric). If it is not a straight line but a curve line, it would be a lognormal distribution.

Figure 4 shows the same conclusions that figure 3. The body of the distribution presents a lognormal behavior (the curved transition point between the two tails is typical of the lognormal distribution), but the tails deviate from the normality, and are closer to a power law behavior (the straight lines in the tails are typical of the power law distribution). Therefore, in this case we have a mixed behavior. This

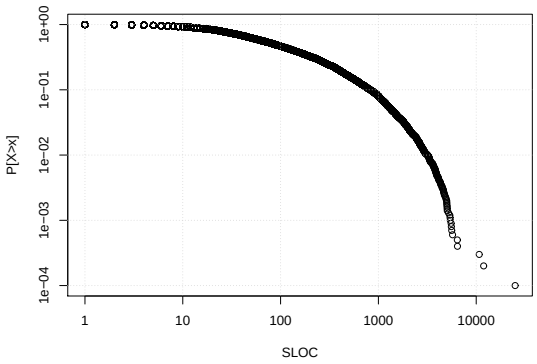


Figure 4. Complementary cumulative distribution function of SLOC. Logarithmic scale.

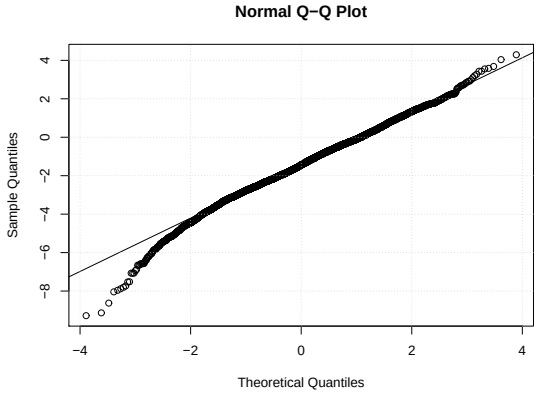


Figure 5. Quantile-Quantile plot for the distribution of the logarithm of the number of comment lines per SLOC.

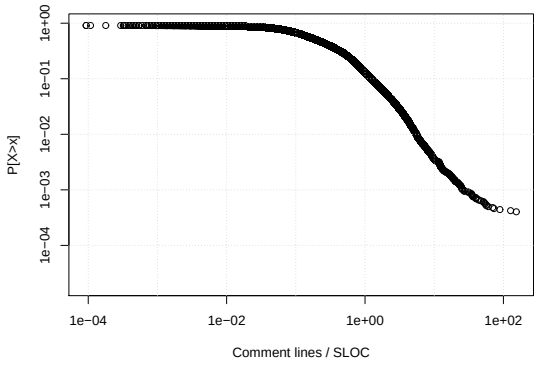


Figure 6. Complementary cumulative distribution function of the number of comment lines per SLOC. Logarithmic scale.

mixed distributions has been found before [14]. It is called a *double Pareto distribution*. The tails for low and high values are straight, and there is a curved transition point (or set of points) where the two straight tails connect. The power law would appear in figure 4 as straight line, and the lognormal distribution would not have straight tails on the extreme values. This behavior has been verified in software before with other samples of source code [7, 6].

As the distribution of the logarithm of the metrics were normal (or more properly, very close to normal), I was able of applying the three sigma rule. In the case of LOC, the standard deviation of the sample was 1.60, and the mean was 4.54. Therefore we can consider those files falling below $4.54 - 3 \cdot 1.60 = -0.23$ and over $4.54 + 3 \cdot 1.60 = 9.34$ in the lognormal distribution to be outliers. These values, transformed back to LOC are $e^{0.6} = 0.77$ SLOC and $e^{9.30} = 11,384$ SLOC. I therefore removed all those files with 0 SLOC. Certainly, I also found files over 11,384 SLOC non automatically generated (this is, at a deep look apparently written by a human developer). But most of files over that value were automatically generated, and the non automatic files were the exception. Therefore I decided to remove those files. This supposed to remove 238 files. The final sample contained then 102,160 files.

I repeated the same plots that in figures 3 and 4, and I did not find any difference between the original and the final sample. So the removal of outliers did not affect the shape of the statistical distribution of the sample. In any case, for instance figure 4 shows clearly that the influence of values was very low (the probability of having files over 10,000 SLOC was around $2 \cdot 10^{-3}$). So I only removed a few points in the right tail, with not very much influence in the distribution.

The properties of the final samples for all the metrics are shown in table 2. As the difference between the mean and the median shows, the distributions were right tailed. When transformed to logarithm, all the distributions became very close to normal.

I also examined some composed metrics: number of blank files per LOC and SLOC, number of comments per LOC and SLOC, and number of comment lines per LOC and SLOC. In the case of these composed distributions, I obtained also double Pareto distributions. For instance, figure 5 shows the Quantile-Quantile plot for the distribution of logarithm of the number of comment lines per SLOC. The distribution is composed for a main body that is normal (along the straight line in the plot), with low and high values tails deviating from normality. The same behavior may be observed on the plot of the complementary cumulative distribution function, in a logarithmic scale. Figure 6 shows that plot; there two straight tails connected by a curved segment. Even in this case the tails are shown more clearly. Again, this is the typical profile of a double Pareto distribution.

Therefore the composed metrics presented also a double

Metric	Mean	Median	Std. dev.	Min.	Max.
SLOC	419	146	715	1	10,733
LOC	495	171	1,440	1	196,429
FUNC	9	2	18	1	791
BLKL	78	27	176	0	20,906
CMTL	59	26	151	0	10,906
CMTN	22	7	70	0	8,748
CYCLO	46	8	49	1	6,802
HLENG	1,854	550	7,060	0	958,333
HVOLU	16,421	3,822	97,583	0	14,610,138
HLEVE	0.0658	0.0297	0.159	0.0000	2.0000
HMD	2,714,000	127,500	$2.38 \cdot 10^7$	0	$2.147 \cdot 10^9$

Table 2. Descriptive statistics of the sample of files written in C (102,160 files).

Pareto distribution. This could mean that there is a correlation among all the metrics. To find out if such correlation exists, I linearly correlated all the metrics. I did not obtain significant correlation coefficients. But when I repeated the procedure with the logarithm of the metrics (ignoring all the files with any of the metrics being zero), I obtained high correlation coefficients among all the metrics.

Table 3 includes all the Pearson coefficients of the correlations among the logarithm of all the metrics. Only the decimal part is shown if the coefficient is not equal to 1. The matrix is symmetrical, so only values under the diagonal are shown. If we take for instance SLOC, we can see that all the correlation coefficients are very high. There are some exceptions, when correlating with blank lines, comments or comment lines. This is a logical conclusion, because those metrics are supposed to be independent (although there is some degree of correlation, for example a large file is likely to contain more comments than a small file). As an example, figure 7 shows the scatter plot of Halstead's length against SLOC, in logarithmic scale. The correlation between the two metrics is clearly shown. There are some points out of the main trend, but precisely because of that the correlation coefficient is not as high as with other pairs.

Therefore, the size and complexity metrics were correlated by means of power laws (linear correlation of logarithms). With only one of the metrics we obtain the same information that we the rest. For instance, SLOC is providing the same information that the rest of size and complexity metrics. I recommend to use only SLOC to characterize the internal attributes of software products. It is easy to collect because there are tools available to count SLOC in many different programming languages.

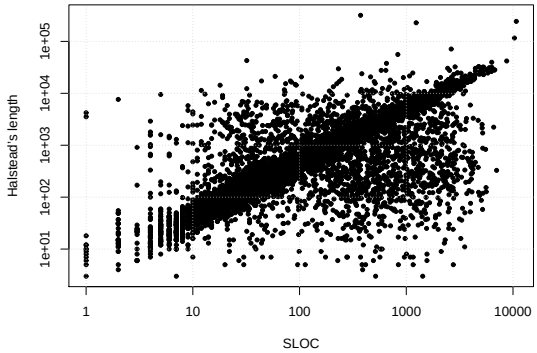


Figure 7. Halstead's length vs. SLOC scatterplot. Logarithmic scale.

6 Towards a theoretical model for software evolution

There exist some proposals of theoretical models to explain phenomena which exhibit double Pareto distributions. Applied to software development, these models could explain how source code files change over time. They could also be applied to other metrics, to explain how they evolve over time, as long as they show a double Pareto distribution. If this finding is verified with other metrics, these models could be the theoretical background for a possible model of software evolution. If not, they would be at least the theoretical background for a model of software writing.

Double Pareto distributions have been found in other fields. For instance, the distribution of the size of the files in a filesystem or tree, follows a double Pareto distribution. For this case, Mitzenmacher [14] proposed the *Recursive Forest File Model*, which is the generalization of others that tried to explain why the size of files follows a double Pareto distribution.

In that model, each tree in the *forest* begins with an initial node of size 1. Then, in each step, a node is randomly

	SLOC	LOC	FUNC	BLKL	CMTL	CMTN	CYCLO	HLENG	HVOLU	HLEVE	HMD
SLOC	1										
LOC	7215	1									
FUNC	6199	7800	1								
BLKL	6552	9136	7909	1							
CMTL	3726	6743	5278	6524	1						
CMTN	5170	7667	6036	7413	8287	1					
CYCLO	6818	8146	7319	7626	5292	6410	1				
HLENG	7532	9536	7591	8613	5284	6862	8522	1			
HVOLU	7502	9534	7506	8630	5284	6845	8423	9985	1		
HLEVE	6672	8879	7633	8303	5702	6752	8825	9166	9108	1	
HMD	7342	9484	7721	8690	5563	6958	8766	9882	9868	9656	1

Table 3. Pearson correlation coefficients between the logarithm of all the metrics. Only the decimal part is shown for values different to 1.

selected. From that node, a new node is created as a child. Each one of the edges is labeled with a value. The size of a file is computed as the sum of the edges going from the initial node to the considered node. This would be easily adapted to the case of a version control system. The initial node would be the initial revision of a file. Each step, would be a new revision. If in a node more than one child appears, then all the nodes but one corresponds to branches parallel to the main trunk. The edges would be changes between revisions, and would be labeled with the size of that change. If a file is deleted, that node disappears from the tree. It would correspond to a revision where the file is deleted.

The model includes also the possibility of adding and removals of new trees to the forest. These new trees would correspond to new files created in some point during the lifetime of the project and added to the version control system.

To obtain the parameters of the model, it would be enough to obtain the parameters of the statistical distribution of the considered metric.

I have not tried yet to make a formal proposal of a model based on the Recursive Forest File Model, but considering the findings of this paper, and the apparent suitability of this model, it seems it would be not difficult to obtain such a model. A model of that kind should be verified against the actual history of the growth of a project. From the verification, I should find out if the model can forecast the evolution of the project.

7 Conclusions and further work

I have analyzed the source code of all the projects stored in the aggregated database of FLOSSMetrics. I obtained the metrics directly querying the MySQL database, from the GNU R tool in order to perform a statistical analysis of the data. The sample contained 102,160 files, with a total size of 32 MSLOC. From this set of files, 52,314 files were written in C. I measured size and complexity, using different metrics (already available in the database), for all these files.

I decided to focus on libre software using the data provided by FLOSSMetrics because of two main reasons: it is easily available in large quantities, and the results of the analysis can easily be checked by other research teams. The decision of considering only the C language has been also practical: most of the tools available to measure code work well with C source code.

All the metrics resulted to be highly correlated by means of power laws. Based on this, and on the fact of the easiness of calculation, I find it interesting to use SLOC to characterize both size and complexity of software products, in any kind of studies. For instance, to study the growth of a software project, in the part regarding the internal attributes of the software, it would be enough to measure only SLOC, to obtain a landscape of the evolution of the size and complexity of the project.

All the metrics were found to follow a double Pareto distribution. This distribution is formed by a lognormal distribution in the main body, and power laws distributions in the tails for high and low values of the distribution. Some composed metrics (such as number of comments per SLOC) presented also double Pareto distributions.

This kind of distributions has been found also for the size of files in a filesystem. Some theoretical models about the growth of file systems have been proposed to explain these cases, whcih have also been used to optimize the download of files from web and FTP servers [2].

The most interesting model we have found is the *Recursive Forest File* model, proposed by Michael Mitzenmacher [14]. It can explain how files change over time, and how they are inserted and removed from a tree of files. Because of its nature, it would be easily adaptable to the case of a source control system, being therefore able of explaining how and why software grows.

In further research, I will try to adapt this model to the

case of a control version system, and to verify it against the actual history of a software project.

References

[1] I. Antoniades, I. Samoladas, I. Stamelos, L. Aggelis, and G. L. Bleris. Dynamical simulation models of the open source development process. In S. Koch, editor, *Free/Open Source Software Development*, pages 174–202. Idea Group Publishing, Hershey, PA, 2004.

[2] P. Badford, A. Bestavros, A. Bradley, and M. Crovella. Changes in Web client access patterns: characteristics and caching implications. *World Wide Web*, 2(1-2):15–28, June 1999.

[3] S. D. Conte. *Software Engineering Metrics and Models (Benjamin/Cummings series in software engineering)*. Benjamin-Cummings Pub Co, 1986.

[4] J.-M. Dalle and P. A. David. The allocation of software development resources in Open Source production mode. Technical report, SIEPR Policy paper No. 02-027, SIEPR, Stanford, USA, 2003. http://siepr.stanford.edu/papers/pdf/02-27.pdf.

[5] M. W. Godfrey and Q. Tu. Evolution in open source software: A case study. In *ICSM '00: Proceedings of the International Conference on Software Maintenance (ICSM'00)*, pages 131–142, Washington, DC, USA, October 2000. IEEE Computer Society.

[6] I. Herraiz. *A statistical examination of the evolution and properties of libre software*. PhD thesis, Universidad Rey Juan Carlos, 2008. http://purl.org/net/who/iht/phd.

[7] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles. Towards a theoretical model for software growth. In *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR)*, pages 21–28. IEEE Computer Society, 2007.

[8] I. Herraiz, D. Izquierdo-Cortazar, F. Rivas-Hernandez, J. M. Gonzalez-Barahona, G. Robles, S. D. nas Dominguez, C. Garcia-Campos, J. F. Gato, and L. Tovar. FLOSSMetrics: Free / libre / open source software metrics. In *Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR)*. IEEE Computer Society, 2009.

[9] I. Herraiz, G. Robles, J. M. Gonzalez-Barahona, A. Capiluppi, and J. F. Ramil. Comparison between SLOCs and number of files as size metrics for software evolution analysis. In *Proceedings of the 10th European Conference on Software Maintenance and Reengineering*, pages 203–210, Bari, Italy, 2006.

[10] S. Koch. Evolution of Open Source Software systems - a large-scale investigation. In *Proceedings of the 1st International Conference on Open Source Systems*, Genova, Italy, July 2005.

[11] M. M. Lehman, J. F. Ramil, and U. Sandler. An approach to modelling long-term growth trends in software systems. In *Internation Conference on Software Maintenance*, pages 219–228, Florence, Italy, November 2001.

[12] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski. Metrics and laws of software evolution - the nineties view. In *METRICS '97: Proceedings of the 4th International Symposium on Software Metrics*, page 20, nov 1997.

[13] M. Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics*, 1(2):226–251, 2004.

[14] M. Mitzenmacher. Dynamic models for file sizes and double Pareto distributions. *Internet Mathematics*, 1(3):305–333, 2004.

[15] G. Robles, J. J. Amor, J. M. Gonzalez-Barahona, and I. Herraiz. Evolution and growth in large libre software projects. In *Proceedings of the International Workshop on Principles in Software Evolution*, pages 165–174, Lisbon, Portugal, September 2005.

[16] G. Robles, J. J. Merelo, and J. M. Gonzalez-Barahona. Self-organized development in libre software: a model based on the stigmergy concept. In *Proceedings of the 6th International Workshop on Software Process Simulation and Modeling (ProSim 2005)*, St.Louis, Missouri, USA, May 2005.

[17] W. M. Turski. Reference model for smooth growth of software systems. *IEEE Transactions on Software Engineering*, 22(8):599–600, 1996.

[18] W. M. Turski. The reference model for smooth growth of software systems revisited. *IEEE Transactions on Software Engineering*, 28(8):814–815, 2002.

Are FLOSS developers committing to CVS/SVN as much as they are talking in mailing lists? Challenges for Integrating *d*ata from Multiple Repositories *

Sulayman K. Sowe
Department of Informatics
Aristotle University
54124 Thessaloniki, Greece
sksowe@csd.auth.gr

Ioannis Samoladas
Department of Informatics
Aristotle University
54124 Thessaloniki, Greece
ioansam@csd.auth.gr

Ioannis Stamelos
Department of Informatics
Aristotle University
54124 Thessaloniki, Greece
stamelos@csd.auth.gr

Lefteris Angelis
Department of Informatics
Aristotle University
54124 Thessaloniki, Greece
lef@csd.auth.gr

ABSTRACT

This paper puts forward a framework for investigating Free and Open Source Software (F/OSS) developers activities in both source code and mailing lists repositories. We used data dumps of fourteen projects from the FLOSSMetrics (FM) retrieval system. Our intentions are (i) to present a possible methodology, its advantages and disadvantages which can benefit future researchers using some aspects of the FM retrieval system's data dumps, and (ii) discuss our initial research results on the contributions developers make to both coding and lists activities.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Open Source Software

Keywords

Free/Open Source Software Development, Software Repositories, Concurrent Versions System, Mailing Lists

1. BACKGROUND

F/OSS developers are not bound to a single project. Even where they are contracted to work in corporate (eg JBoss), or foundation (Apache) projects, they still have the freedom to participate in other projects or communities of interest. They code, take part in discussions in various mailing lists and forums, and occasionally participate in agile-styled project's sprints. Along the way they leave a trail of experience, wealth of knowledge and skills associated with their art. This may be in the form of large and small bits of code, coding ethics and guidelines, documentations, etc. Participants in various F/OSS projects use tools (Versioning Systems (CVS/SVN), mailing lists, Bug tracking systems, etc.) to enable the distributed and collaborative software development process to proceed. These tools serve as repositories which can be data mined to understand *who* is involved, who is talking to *whom*, *what* is talked about, *how much* some one contributes in terms of code commits or email postings. Such information provides insights into the nature of collaboration in the projects concerned. Repositories of some F/OSS projects have extensively been used to better understand the contribution of developers [12, 6], trends and inequality in posting and replying activities in Apache and Mozilla [10], KDE [8], Debian developer and non-developer lists [15], FreeBSD [4], to mention a few. Empirical research in these areas is aided by the fact that F/OSS data is widely and freely available in various repositories in various formats [11, 7]. However, there are problems associated with aggregation and extraction of F/OSS data [2, 14]. A trend in F/OSS research is the use of data stored in a repository of repositories or *RoRs* (e.g. FLOSSMole and FLOSSMetrics)[1], in which data from many and varied projects are brought under one umbrella so that researchers can have easy access [14]. One major advantage of RoRs is that they offer an aggregated mixture of metadata in various formats, allowing researchers to concentrate more on there analysis than data scouting. This research benefits from such RoRs; the FM retrieval system (`http://fm3.libresoft.es/retrieval_system/`) of the FLOSSMetrics project.

*This research is partially sponsored by the FLOSSMetrics Project (Ref. No. FP6-IST5-033547), `http://flossmetrics.org/` and SQO-OSS project (Ref. No. FP6-IST-5-033331),`http://www.sqo-oss.eu/`

[1]http://ossmole.sourceforge.net/ and http://flossmetrics.org/

F/OSS projects are different in their organizational structures as well as their unique way of doing things. However, certain aspects are fundamental to all projects. Source configuration management (SCM), of which CVS or SVN is a part, is a *de facto* tool used to coordinate and view the coding activities of software developers, manage software builds and releases, and other software development related activities. Mailing lists, on the other hand, are the main communication channels [15]. Many important aspects of a project are negotiated in developer lists: software configuration details, the way forward and how to deal with future requests, how tasks are distributed, issues concerning package dependencies, scheduling online and off-line meetings, etc. For a developer to keep abreast with developments in a project, committing code to SVN alone is not sufficient. S/he needs to participate in the respective lists, communicate his ideas, and engage with colleagues. However, due to the volunteering nature of F/OSS development [9], developers are free to choose what to work on, and where to contribute and channel their efforts. Thus, a comprehensive investigation of developers must not only concentrate on their code contribution but also revisit their activities in other project's media (e.g. developer mailing lists) and compare and contrast their quantitative and qualitative contributions.

1.1 The purpose of our research

F/OSS researchers study and report developers coding activities in CVS [10, 8, 4] or change logs [1] separately from their mailing lists activities. Important as these studies and their findings are, we conjecture that not all the developers who commit or make changes to a project's source repository also participate in developer mailing lists. In order to *fill this gap in F/OSS research*, this study investigates the concurrence or simultaneous occurrence of F/OSS developers in both SVN and developer mailing lists. That is, we find out if F/OSS developers are coding through commits in SVN as much as they are "talking" in developer mailing lists.

Our understanding of the F/OSS development process informs us that in many projects, a small number of talented core developers or "cod gods" are busily tinkering with code to produce good and usable software for the rest of the community. We also know the contribution these code gods make to discussions in mailing lists; they interact with other software developers and users, the keep abreast with project activities and monitor the what goes on in there projects. Little or no research has attempted to correlate developers commits activities with their corresponding mailing lists activities, either within the same project or across projects. Active mailing lists is a proxy of project success [5, 3]. However, involvement in mailing lists should not only be limited to non-developers. The presence of project's leads, core and active developers in mailing lists has a profound effect on the way individuals within and outside the project see the commitment of the most influential members in the project. For software companies and private enterprises, their presence in lists may indicate that software support activities are not only available from ordinary users, but also comes form individuals behind the software and project. This argument leads to the formulation of the following hypothesis:

H0 (null): FLOSS developers contribute equally to code repository and mailing lists, with alternative:

H1: FLOSS developers contribute more to code repository than mailing lists.

In what follows, we investigate the hypothesis and draw conclusions. Our data comes from SVN commits and mailing lists archives of *fourteen* F/OSS projects from the FM retrieval system. First, in Section 2, we present the methodology used in this research, followed by an analysis of our preliminary results in Section 3. Implications of our findings, limitations, and work in progress concludes this paper in Section 4. The observations we have made so far may provide some salient issues to be discussed and fine-tuned with workshop participants.

2. RESEARCH METHODOLOGY

Our research methodology aims to overcome challenges associated with investigating the simultaneous occurrence of developers in SVN and mailing lists. This has to do with difficulty in (*identification*) making sure that the developer making SVN commits in a project is the same individual posting to the developer mailing list(s) of the same project. An outline of the methodology (figure **1**) shows the FM retrieval system as our data set choice. In addition to mailing lists and CVS/SVN data dumps, the system provides many attributes of a project including project's description and website, SCM and mailing lists urls. One important use of the SCM urls is to allow researchers to locally checkout and browse projects' repositories and view and analyze change log data. Researchers can freely download and use available SVN and mailing lists data dumps. Bug databases of FM projects are currently being processed. For a detailed specification, design, and description of the FM database, refer to the FLOSSMetrics project's work-package 3 deliverable (3.1, 3.2)[2]. Figure **2** shows three tables from the FM database that we used to obtain SVN and mailing lists data.

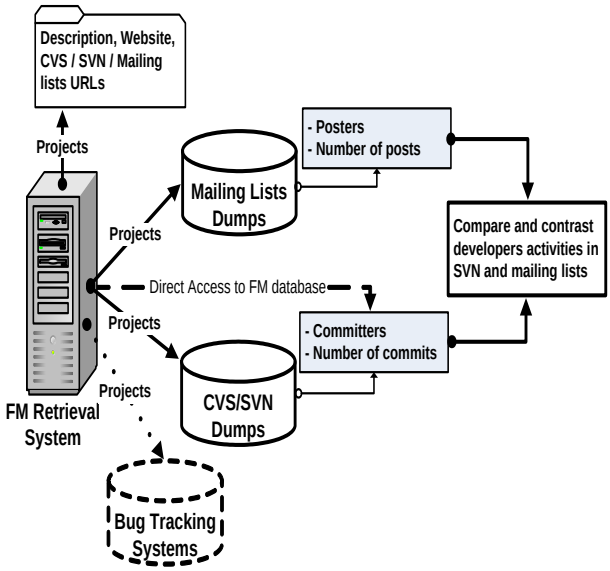


Figure 1: Outline of research methodology

[2] `http://flossmetrics.org/sections/deliverables/WP3`

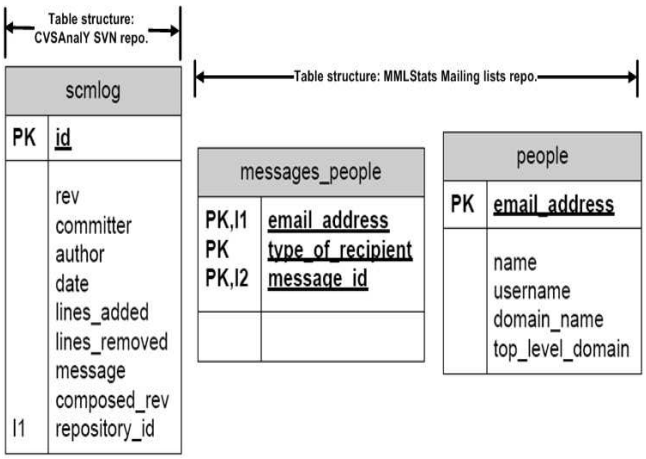


Figure 2: FM's SVN and Mailing list tables schema

As of 31st May, 2008, the FM beta version contained 60 projects with both analyzed data from source code management and from mailing lists repositories. From these we randomly selected 14 projects. Our choice of projects was guided by two selection criteria:
- projects should cover as many domains as possible, and
- developer contribution in terms of commits and postings should vary as much as possible.

With this criteria we hoped avoid selecting and studying only "successful" projects with large developer contribution and being bias towards one domain.

2.1 SVN Data

The CVSAnalY[3] database dumps provided by the FM retrieval system contains tables with SVN actions and information on committers. However, instead of downloading and using the SVN dumps, we used a python script to access the FM database, as a test and an alternative (see figure **1**). Each project in the FM database can have one or more SVN dumps. For each project, we extracted the SVN committers and the number of commits they made. These two values act as fields in an mysql table (*commits*) for committer identification in each project (see figure **3**). Table **1** shows descriptive statistics of the SVN data in the fourteen projects studied. For each project the total number of SVN

Table 1: Projects by Commits before analysis

Project	Nĉ	Mean	Median	Std. Dev.	Skew.	Max. Commits	Total commits
activemq	13	153.54	23.00	287.974	2.085	854	1996
ant	62	349.19	95.50	607.249	2.821	2908	21650
apr	132	191.56	55.00	318.549	2.230	1277	25286
beehive	13	330.54	258.00	331.582	1.475	1015	4297
ekiga	3	1010.33	459.00	1295.219	1.568	2490	3031
felix	12	58.92	30.00	85.896	1.758	260	707
gdm	24	92.67	20.50	203.638	3.265	894	2224
gedit	18	77.78	4.50	154.890	2.564	587	1400
ibatis	5	170.40	157.00	172.575	1.504	458	852
libsoup	4	170.25	171.00	190.241	.000	335	681
nautilus	171	94.01	13.00	210.142	3.112	1202	16075
openejb	4	583.75	339.50	767.955	1.243	1647	2335
Spamassassin	14	1106.07	212.50	1278.926	.741	3120	15485
turbine	24	107.92	77.50	128.307	1.786	540	2590

committers (Nĉ), the mean commit per committer, the total number of commits made by Nĉ developers, and other relevant statistics, are shown.

[3]http://cvsanaly.tigris.org/

2.2 Mailing Lists Data

The MLStats[4] database data dumps provided by the FM retrieval system contains one or more mailing lists archives of a particular project. Structurally, the data is a dump of the retrieval system's database ("*fm3_activemq_mls*"). We downloaded **.sql* files dumps of each project and extracted data contained in two tables:
- *messages_people* table (**email_address**, type_of_recipient ('**From**','To','Cc'),...), and
- *people* table (**email_address**, **name**, **username**,...).

This information acts as fields in two mysql tables for mailing lists posters identification in each project (see figure **3**). Table **2** shows, for each project, the total number of posters

Table 2: Projects by Posts before analysis

Project	Np̂	Mean	Median	Std. Dev.	Skew.	Max. Post	Total posts
activemq	1548	8.03	2.00	65.898	34.813	2487	12430
ant	7301	8.67	3.00	52.725	32.967	2704	63312
apr	1060	17.47	2.50	70.120	10.842	1433	18514
beehive	272	6.57	2.00	16.970	5.874	152	1787
ekiga	525	7.93	2.00	47.647	19.774	1040	4163
felix	319	21.52	3.00	102.587	10.947	1283	6866
gdm	640	3.56	1.00	21.666	18.982	484	2281
gedit	471	4.11	1.00	15.543	12.085	266	1935
ibatis	1406	9.13	3.00	40.500	15.507	784	12830
libsoup	26	6.23	2.50	8.262	2.672	37	162
nautilus	2091	16.81	4.00	117.340	32.373	4719	35150
openejb	77	13.88	4.00	34.549	5.897	271	1069
spamassassin	4658	28.91	6.00	148.806	16.883	4248	134649
turbine	1874	21.01	6.00	70.111	10.700	1182	39377

(Np̂), the mean post per poster, the total number of posts made to the project's mailing list by Np̂ developers, and other relevant statistics, are shown.

2.3 Identifying Developers

In CVS or SVN and change logs, an individual is simply identified as a "Committer" or an "Author" of one or more commits. Mailing lists participants, on the other hand, can be identified by means of message identifiers like "From:" in email headers [13].

After extracting data of SVN committers and mailing lists posters, we proceeded with identifying developers as shown in figure **3**.

We queried the three tables to obtain data for our analysis. With this method, we were able have fairly accurate identity of individuals and their SVN and mailing lists contribution.

3. INITIAL RESULTS & ANALYSIS

Table **3** shows the number of developers (N) who made SVN commits and posted information to the project's mailing lists. Comparing tables **3** and **1**, in eight out of the fourteen (57.14%) projects studied, all the SVN committers also participated in mailing lists discussion. In four projects over 90% and in two projects over 80% of the committers participated in lists. As shown by the sum of commits and posts in tables **3**, developers in each project, with the exception of the *ibatis* and *turbine* projects, made more commits than posts. The mean commit per developer in each project, with

[4]`http://tools.libresoft.es/mailing_list_stats`

Table 3: Statistics on Developers who made both commits and posts

Project	Commits						Posts					
	N	Sum	Maximum	Mean	Median	Std. Deviation	N	Sum	Maximum	Mean	Median	Std. Deviation
activemq	13	1996	854	153.54	23.00	287.974	13	807	380	62.08	5.00	109.439
ant	62	21650	2908	349.19	95.50	607.249	62	10026	2704	161.71	21.00	422.599
apr	129	24857	1277	192.69	56.00	320.824	129	11319	1433	87.74	22.00	173.654
beehive	12	3282	1015	273.50	239.50	271.666	12	202	104	16.83	6.50	28.565
ekiga	3	3031	2490	1010.33	459.00	1295.219	3	1228	1040	409.33	184.00	553.539
felix	12	707	260	58.92	30.00	85.896	12	457	223	38.08	15.50	61.745
gdm	20	2063	894	103.15	16.00	222.023	20	467	227	23.35	2.00	53.664
gedit	17	1399	587	82.29	5.00	158.431	17	334	266	19.65	2.00	63.659
ibatis	4	695	458	173.75	114.50	199.085	4	750	740	187.50	4.50	368.343
libsoup	4	681	335	170.25	171.00	190.241	4	68	37	17.00	13.50	15.684
nautilus	168	15968	1202	95.05	13.00	211.803	168	12891	4719	76.73	10.00	384.816
openejb	4	2335	1647	583.75	339.50	767.955	4	84	74	21.00	4.00	35.384
spamassassin	14	15485	3120	1106.07	212.50	1278.926	14	7235	2834	516.79	23.00	960.613
turbine	24	2590	540	107.92	77.50	128.307	24	15390	2364	641.25	224.00	825.384
Total	486	96739	3120	199.05	29.50	434.670	486	61258	4719	126.05	14.00	402.856

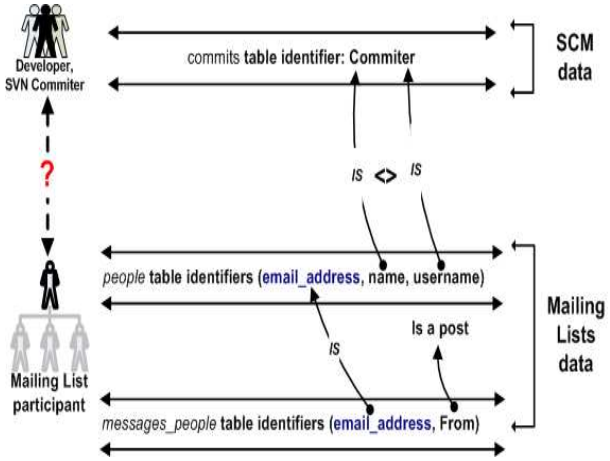


Figure 3: Identifying developers from multiple repositories (CVS and Mailing Lists)

the exception of the *turbine* project, are also greater than the mean post per developer.

3.1 Distribution of Commits and Posts

The box plots in figure **4** show the distributions of commits and posts for every project. The domination of commits over posts is evident in most of the projects. A comparison

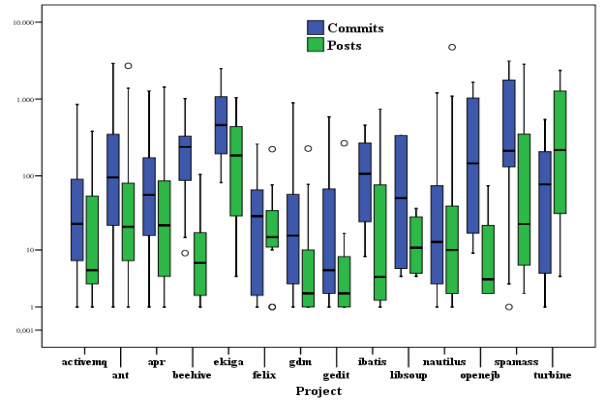


Figure 4: Distributions of commits and posts for every project. *Y-axis in a logarithmic scale*

of the distributions of commits and posts for all developers together is shown in Figure **5**. The domination of SVN commits, with larger means of commit per developer, over mailing lists posts is evident.

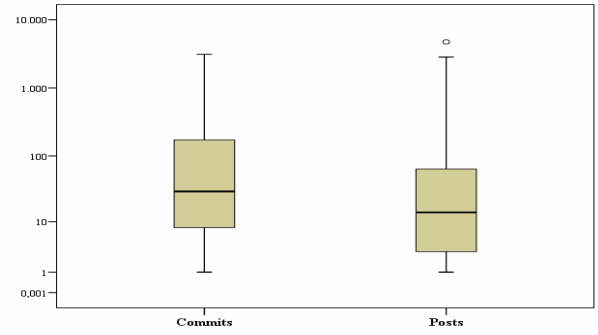


Figure 5: Distributions of commits and posts for all developers. *Y-axis in a logarithmic scale*

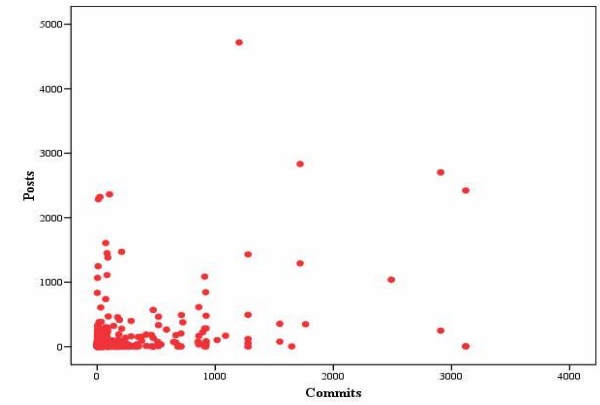
3.2 Relationship Between Commits and Posts

We used correlation between commits and posts to study how developers activities in SVN and mailing lists are related. The scatter plots in figure **6** shows the correlation between Commits and Posts in all projects, in two different scaling dimensions.

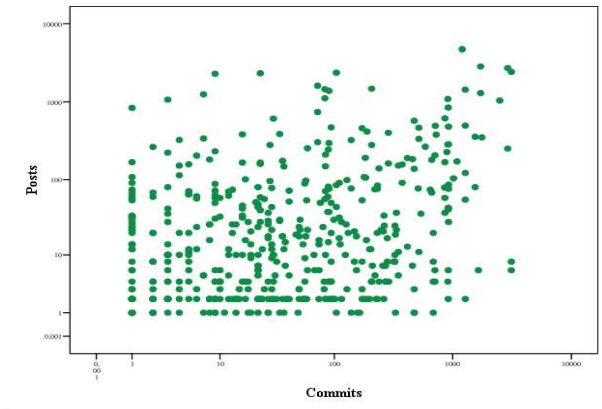
Since the distributions of both variables (commits and posts) are highly skewed and different from normal (kurtosis commits=20.150, std. dev.=434.670; kurtosis posts=47.720, std. dev=402.856), we use the nonparametric coefficient of Spearman which is based on ranks. For each project separately and for all projects together, table **4** gives the Spearman's coefficient (ρ) and their significance (p-value). Note that the significance is dependent on the size of the sample. Projects showing statistical significance ($p<0.05$) are are marked with an asterisk(*).

3.3 Developers Contribution

In order to measure how much developers contributed in terms of commits and posts, we tested the difference between commits and posts for each project and for all the fourteen projects together. We use the Wilcoxon signed ranks test for two related samples. For each project separately, and for all projects together, table **5** shows the significance (p-value) of the test. Statistically significant differences are



(a) for all projects in linear scale



(b) for all projects in logarithmic scale

Figure 6: Correlations between commits and posts

Table 4: Commits-Posts Correlation measures

Project	N	Spearman's ρ	Significance (p-value)
activemq	13	**0.612***	**0.026**
ant	62	**0.294***	**0.020**
apr	129	**0.283***	**0.001**
beehive	12	**0.317**	**0.315**
ekiga	3	**1.000***	**0.000**
felix	12	**0.173**	**0.591**
gdm	20	**0.350**	**0.130**
gedit	17	**0.528***	**0.029**
ibatis	4	**0.000**	**1.000**
libsoup	4	**0.949**	**0.051**
nautilus	168	**0.123**	**0.113**
openejb	4	**0.738**	**0.262**
spamassassin	14	**0.171**	**0.559**
turbine	24	**-0.197**	**0.357**
All projects Together	486	**0.266***	**0.000**

considered those having $p<0.05$. In case of significant difference, from the mean ranks we can understand whether commits are generally more than posts. This is denoted by "*commits>posts*", otherwise we write "*commits<posts*". In projects where there is no statistical difference ($p>0.05$) we write "*commits=posts*".

Table 5: Difference in developers' contributions

Project	N	Significance (p-value)	Type of difference
activemq	13	**0.124**	**commits=posts**
ant	62	**0.000**	**commits>posts**
apr	129	**0.000**	**commits>posts**
beehive	12	**0.002**	**commits>posts**
ekiga	3	**0.109**	**commits=posts**
felix	12	**0.424**	**commits=posts**
gdm	20	**0.006**	**commits>posts**
gedit	17	**0.007**	**commits>posts**
ibatis	4	**0.715**	**commits=posts**
libsoup	4	**0.109**	**commits=posts**
nautilus	168	**0.086**	**commits=posts**
openejb	4	**0.068**	**commits=posts**
spamassassin	14	**0.035**	**commits>posts**
turbine	24	**0.012**	**commits<posts**
All projects Together	486	**0.000**	**commits>posts**

4. DISCUSSION AND CONCLUSIONS

In this paper we have investigated whether F/OSS developers are committing more to SVN than they are posting to mailing lists. This involves tracking their activities in two or multiple repositories and studying their quantitative contribution. This kind of research is made difficult because of the problem associated with the identification of developers in both repositories. The FM retrieval system's table schema has fields attributes which enabled us to overcome this research obstacle so that we are able to study the simultaneous occurrence of developers and measure their contributions in multiple repositories (SVN and mailing lists). The methodology presented in this paper is a possible means to leverage problems associated with empirical F/OSS research in this area.

The conclusion of our research supports our hypothesis. *H0 (null)*, FLOSS developers contribute equally to code repository and mailing lists. The null hypothesis is that the number of commits and the number of posts have the same distribution (ρ=0.000 for all the 486 committers who are also posters in the fourteen projects). Alternatively (*H1*), FLOSS developers contribute more to code repositories than mailing lists. The distributions of the two variables (commits and posts) are different for each project (with commits dominating) as shown by the various plots (see figures 4- 6). Furthermore, looking closer at the cases where the difference is not significant, we can see that the number of commits is greater. For example, the *nautilus* project has p=0.086 providing some evidence of difference at the 0.10 level and, domination of commits. The only exception was observed in two projects (*turbine* and *ibatis*), where the number of posts are greater than the number of commits. One explanation for this deviation could be that in both projects the prolific committer who is also the most prolific poster had exceptionally high commits (540 commits for the turbine project and 458 commits for the ibatis project) and posts (2364 posts for the turbine project and 740 posts for the ibatis project)relative to the rest of the community. But while the contribution of these prolific developers might skew the commits and posts in the projects, there could be other explanations. For example, both (ibatis, turbine) are Apache projects. Perhaps there is an underlying cause due to the structure or nature of Apache projects?

However, a number of factors or questions remains unanswered in this conclusion that we have not yet investigated. For example: **At commit level**: - What *kinds* of commits are the developers making? are they modifications, dele-

tions, additions? - What *types* of commits are the developers making? are the commits source code related, documents? - Time-stamp, when did a developer make those commits? **At mailing lists level**: - Is there a way we can we consider postings to developers lists only from the FM mailing lists dumps? - What category of information are the developers posting? - What about 'replies'? are there developers who are just posting project or software package information without replying to postings others made to the lists? Answers to some of these questions may help us to have a qualitative insight into the developers contribution to both repositories. And support observations we have made so far.

4.1 Few things we want to do

Since we have studied only those developers who are in both repositories, there is a tendency that we have missed some high prolific committers who may play a vital role in the project. We intend to look at other contributors with substantial commits and find out why they have no postings in their project's developer list(s). In our methodology, manual scan was necessary to remove some duplicates. The results of our SQL query sometimes yields results which require manual intervention. We are investigating how this methodology can be automated, perhaps by extending the functionality of CVSAnaY and MLStats data extraction and analysis tools.

4.2 Ongoing & future work

1. As the FM retrieval system database continues to grow, we are extracting data from a number of projects which have SVN and mailing lists dumps available to further investigate the trends we have observed.

2. Using information available at the FM retrieval system we can extend the analysis of developers participation in multiple repositories by including data from each project's change log. We have observed that the top "commits by author" in the change logs of each of the projects studied happens to be in top committers and posters in our data.

5. ACKNOWLEDGMENTS

We wish to extend sincere gratitude to all FLOSSMetrics and SQO_OSS partners who offered valuable comments and suggestions. In particular; Santiago Dueñas(FLOSSMetrics), Georgios Gousios(sqo-oss), etc.

6. REFERENCES

[1] K. Chen, S. R. Schach, L. Yu, J. Offutt, and G. Z. Heller. Open-source change logs. *Empirical Softw. Engg.*, 9(3):197–210, 2004.

[2] M. S. Conklin. Beyond low-hanging fruit: Seeking the next generation in floss data mining. In E. Damiani, B. Fitzerald, W. Scacchi, M. Scott, and G. Succi, editors, *IFIP International Federation for Information Processing, Open Source Systems.*, volume 23, pages 261–266. IFIP, Boston: Springer., 2006.

[3] K. Crowston, A. Hala, and J. Howison. Defining open source software project success. In *Proc. of International Conference on Information Systems, ICIS 2003*, 2003.

[4] T. T. Dinh-Trong and J. M. Bieman. The freebsd project: A replication case study of open source development. *IEEE Transactions on Software Engineering*, 31(6):481–494, 2005.

[5] K. Fogel. *Producing Open Source Software: How to Run a Successful Free Software Project.* CreativeCommons, 2005.

[6] G. Gousios, E. Kalliamvakou, and D. Spinellis. Measuring developer contribution from software repository data. In *MSR '08: Proceedings of the 2008 international workshop on Mining software repositories*, pages 129–132. ACM, 2008.

[7] A. E. Hassan. *Mining Software Repositories to Assist Developers and Support Managers.* PhD thesis, School of Computer Science, Faculty of Mathematics, University of Waterloo, Ontario, Canada., 2004.

[8] G. Kuk. Strategic interaction and knowledge sharing in the kde developer mailing list. *MANAGEMENT SCIENCE 2006 52: 1031-1042.*, 52:1031–1042, 2006.

[9] M. Michlmayr. *Quality Improvement in Volunteer Free and Open Source Software Projects: Exploring the Impact of Release Management.* PhD thesis, University of Cambridge, 2007.

[10] A. Mockus, R. Fielding, and J. Herbsleb. Two case studies of open source software development: Apache and mozilla. *Transactions on Software Engineering and Methodology.*, 11(3):1–38, 2002.

[11] G. Robles. *Empirical Software Engineering Research on Libre Software: Data Sources, Methodologies and Results.* PhD thesis, Dept. of Informatics. Universidad Rey Juan Carlos, Madrid, Spain., 2005.

[12] G. Robles and J. Gonzalez-Barahona. Contributor turnover in libre software projects. In E. Damiani, B. Fitzgerald, W. Scacchi, M. Scotto, and G. Succi, editors, *IFIP International Federation for Information Processing, Open Source Systems*, volume 203, pages 273–286. Springer,Boston, 2006.

[13] S. K. Sowe, L. Angelis, and I. Stamelos. Identifying knowledge brokers that yield software engineering knowledge in oss projects. *Information and Software Technology*, 48:1025–1033., 2006.

[14] S. K. Sowe, L. Angelis, I. Stamelos, and Y. Manolopoulos. Using repository of repositories (rors) to study the growth of f/oss projects: A meta-analysis research approach. In *Open Source Development, Adoption and Innovation*, volume 234/2007 of *IFIP International Federation for Information Processing*, pages 147–160. Springer Boston, August 2007.

[15] S. K. Sowe, I. Stamelos, and A. Lefteris. Understanding knowledge sharing activities in free/open source software projects: An empirical study. *Journal of Systems and Software*, 81(3):431–446., 2008.

On the Approximation of the Substitution Costs for Free/Libre Open Source Software

Deliverable ID: D5.1

Version: 1.0

Deliverable: D5.1

Title: **On the Approximation of the Substitution Costs for Free/Libre Open Source Software**

Authors: K. Haaland, I. Stamelos, R. Ghosh, R. Glott

Executive Summary:

In this paper we describe our work in progress towards approximating the substitution costs for FLOSS. Substitution cost is the monetary value of the effort necessary for implementing a FLOSS application from scratch in a software company. We describe our approach for estimating substitution cost which is based on building generic estimation models for the modern software industry. For this purpose we use the latest version of ISBSG, and apply our models on a subset of projects from a Debian distribution. This yields an approximation of the total substitution costs for this collection of FLOSS applications. We report on various problems, limitations and issues related to the data, the model precision, and the currently available FLOSS project data. However, the anticipated arrival of more coherent FLOSS databases is expected to alleviate many of these issues, and more precise calculations will become feasible, using the same methodology as outlined in this paper.

Keywords

Free / Libre Open Source Software, FLOSS, ISBSG, substitution cost, DEBIAN, estimation by analogy, regression

1. Introduction

A wealth of successful software applications has been brought forward by Free/Libre Open Source Software (FLOSS); the new software development paradigm based on volunteer participation and open development processes. FLOSS plays an increasingly important role in the ICT sector in particular, and the world economy in general. It is therefore interesting to attempt to estimate the economic impact of FLOSS.

In this paper we report our efforts on providing an informed estimate ([1]) of the substitution costs for a large portion of a Free/Libre Open Source Software code base. Substitution costs refer to how much it would have cost to build the same software from scratch entirely within a single firm in a proprietary software development model. Calculating the substitution costs for a given set of FLOSS applications is one way of assigning a Euro value to the production and effort represented by these applications.

We use the latest version of ISBSG dataset (www.isbsg.org) to build the estimation models, and apply our models on a subset of projects from the DEBIAN release 3.1 (also known under the name Sarge), producing an approximation of their substitution cost. We wish to independently generate as many models as possible, in order to deal with the inherent prediction uncertainty involved in building a cross-organizational model ([2]). In this paper we report results from two well-known estimation methods, namely Ordinary Least Squares Regression (OLS) and Estimation by Analagy (EbA). We produce two models with each method and compare their results.

Our study is conducted in the context of the FLOSSMetrics project (http://flossmetrics.org). The objective of FLOSSMetrics is “to construct, publish and analyze a large scale database with information and metrics about FLOSS software development coming from several thousands of software projects, using existing methodologies, and tools already developed”. Since the database is still under construction (October 2008), we use a currently available dataset of Debian 3.1 to draw preliminary conclusions about the FLOSS substitution costs.

The rest of the paper is organized as follows: Section 2 develops and describes methodology for substitution cost estimation, i.e. the prerequisites that an estimation model should fulfill in order to be used for the calculation of FLOSS substitution cost. Section 3 describes the preparation of ISBSG and DEBIAN subsets, while Section 4 provides a brief description of model generation and reports preliminary results. Finally Section 5 contains preliminary discussion and conclusions, as well as future work.

2. Methodology for substitution cost estimation

Our aim is to calculate substitution costs for a set of FLOSS projects. We will do that by estimating the substitution cost for each project separately and will proceed by producing an estimate for the entire project set. Estimation for each project separately is possible through the application of a parametric estimation model, i.e. a model that will take into account project parameters such as size, application domain, elapsed time etc. Such an approach has been already taken by Boehm in 1981 with the COCOMO model, a within company estimation model, based on 63 projects which are dated from 1970 to 1980 [1]. Later FLOSSIMPACT applied COCOMO on Debian 3.1 [3] .

Our aim is to develop a cross-organizational estimation model that is representative of modern software industry. Such model seems more suitable than COCOMO 1981 model for our purposes because it is reasonable to presume that FLOSS would not be substituted by a single company, since it is known that different companies have different productivity. In addition, productivity in recent years is different (typically higher) than in the '70s.

We hereby provide a comprehensive list of the characteristics, which such model should have according to our current understanding of the problem:

1. The model should be representative of the modern, global software industry. The model should capture productivity levels that would support our requirement for estimating costs of a typical, average productivity software company. We will use the ISBSG dataset for generating estimation models.
2. The model should be based on some measure of physical size. Functional sizing (e.g. Function point analysis) is not practiced in FLOSS, and it would be infeasible to measure the functional size of thousands of FLOSS projects, given that no automation tools are available for such task at the moment. We chose to use Source Lines of Code as a measure of physical size because they are readily available for FLOSS projects. SLOC have been used also in the study that can be found in FLOSSIMPACT [3]. However limitations of SLOCs are already known, so we need to take any possible precaution for their use in the model. In particular, we should keep in mind that SLOC measurements may contain a lot of noise.
3. The model should be based on closed source projects similar to those found in FLOSS. In particular, it should be based on projects that are similar to the FLOSS projects for which it will be used to produce substitution costs. Project attributes that define project identity, and potentially affect productivity and therefore costs, are the project application type, development platform, language type, size etc. The methodology should build generic estimation models that take into account these attributes, then, while calculating substitution costs, carefully chosen values to characterize FLOSS projects should be used.
4. The estimation model should produce interval estimates to account for data uncertainty. Given that the model will be calibrated on multi-organizational project data and will be applied on hundreds or thousands of projects developed by communities of volunteer programmers, a range of possible cost values should be produced. Ideally, the estimate for each FLOSS project and for the entire set of FLOSS projects should be a range of values along with a probability distribution.

5. Because of the inherent uncertainty in this estimation context, we should build more than one model. This is a typical precaution in software cost estimation. The precision of the models should be first assessed and then they should be applied to the FLOSS projects under study to produce a variety of estimates. Such estimates should then be compared in order to (a) produce the final estimate and (b) assess the overall precision of the calculated substitution cost.

Having defined the prerequisites of the estimation models we now proceed with defining the steps for generating the models. The various stages of the proposed methodology are based on standard empirical software engineering practice. However, we will follow the approach that has been taken in [4]. Zhou and Leung have applied a similar methodology on the code base of ANT (an Apache project) to study the relationship between structural metrics and unit testability in object oriented systems.

3. Dataset Preparation

We base our study on two datasets, namely the ISBSG dataset and Debian 3.1. The ISBSG dataset Revision 10 (called ISBSG R10 in the rest of the paper) consists of only closed source projects, and represents the modern software industry. ISBSG is a non-profit organization that collects data on software projects from all over the world. We believe that ISBSG is the most reliable source for building a cross-organizational software cost estimation model [2]. Our training set is a subset of ISBSG R10, which contains 443 projects measured with SLOC. Additional ISBSG categorical project attributes that are meaningful for FLOSS projects as well are language type, resource level, and development type and platform (see www.isbsg.org).

Our first concern was to validate SLOC as size metric for our data. ISBSG organization provides guidelines for using the data in ISBSG R10. Because they have not validated SLOC values they advise not to use SLOC counts. We overcame this issue by (a) inspecting SLOC counts and performing a sanity check, leading to the adjustment of some evidently wrong numbers, (b) analyzing the correlation with Function Point (FP), which is the major size metric in ISBSG, and (c) referring to other papers that have already used SLOC related information from ISBSG and reached meaningful results (e.g. [5], [6]).

After some data cleaning (projects with low, i.e. C or D, ISBSG data quality rating have been excluded) and outlier analysis based on productivity measured in SLOC/manhour, we produced a dataset of 395 projects (ISBSG-SLOC-all subset).

In order to investigate the validity and quality of SLOC as size metric in ISBSG R10, we built regression models with SLOC as independent and Summary Work Effort as dependent variable in ISBSG, after transforming them on a logarithmic scale (variables LNSLOC and LNEFFORT respectively, while LNFP stands for the logarithmic transformation of variable Functional Size in ISBSG). A linear regression model with LNSLOC as independent and LNEFFORT as dependent variable produced an adj-R2 equal to ~0.57, showing that there is a good correlation between the two variables. Another linear regression model, with LNSLOC as independent and LNFP as

dependent variable produced an adj-R2 equal to ~0.6, showing that there is a good correlation between the project functional and physical size as well. Interestingly, a regression model with LNFP as independent and LNEFFORT as dependent variable produced an adj-R2 equal to ~0.50, showing significant, but weaker than SLOC, correlation between Function Points and project effort.

Based on the results of the correlation analysis and the use of SLOC in ISBSG by other researchers, we decided to proceed with our study, using SLOC as a proxy for the physical size of both closed and open source projects. The ISBSG-SLOC-all 395 project dataset consists of quite heterogeneous projects, written in various languages. It is difficult to generate an accurate effort estimation model under these conditions, so we devise two further subsets. One subset, namely ISBSG-SLOC-ND, consists of only 176 “New Development” projects written in various languages. Another subset, namely ISBSG-SLOC-C, consists of 48 ISBSG-SLOC-all projects all written primarily in the C language.

Debian 3.1 consists of 6173 projects in total. We chose an old version of Debian because the majority (57%) of the projects in Debian 3.1 are written in C language, which helps reducing the uncertainty for our estimates. We further developed a subset of 5157 projects that were of comparable size to those found in the two ISBSG training datasets. For the descriptive statistics, see Table 1.

Table 1: Descriptive statistics for the three datasets.

	ISBSG-SLOC-ND	**ISBSG-SLOC-C**	**DEBIAN-1**
N	176	48	5157
Total SLOC	12467849	1852572	101886869
Max	1900000	334800	333936
Min	100	400	339
Mean	70840.05	38595.25	19757
Median	22638	17716.50	5012
Stdev	198819.60	64345.11	39671

Overall the ISBSG and Debian datasets’ similarities and differences are:

1. Similarities

   – both are of “cross-company” nature, OSS projects are developed by a multitude of developers

   – both are representative of world-wide

2. Differences

– closed source vs. open source, different developer motivation, team sizes, release rate, development process, project deliverables (documentation, ...)

– differences in descriptive statistics (should always be minimized).

4. Building the Estimation Models

This section provides details on how far we have gone with building estimation models. Up to now regression models (OLS), and analogy based estimation (EbA is probably the most widespread methods for effort estimation) have been applied. In the future we expect to explore more methods, like categorical regression models, analogy combined with regression, and machine learning methods.

4.1 OLS Model

OLS models have been produced with the use of the statistical software package STATA®. In creating the different datasets, an outlier analysis was performed. ISBSG categorical variables have been handled through dummy variables. The adj-R2 for ISBSG-SLOC-ND was ~0,78 and for ISBSG-SLOC-C ~0,66. Table 2 reports precision figures (MMRE and PRED25) for the two models.

Table 2 Summary results for OLS estimation models.

ISBSG-SLOC-ND		ISBSG-SLOC-C	
MMRE	PRED25	MMRE	PRED25
58,26%	28,41%	55,47%	27,08%

MMRE stands for Mean Magnitude of Relative Error, and is a common measure of precision in software engineering. MRE for a single prediction is given by the formula in equation 1 (y_A is the actual project effort and y_E is the estimated effort).

$$MRE = \frac{(y_A - y_E)}{y_A} \quad (1)$$

PRED25 refers to the amount of projects that are estimated with less than 25% error. Typically in software cost estimation, accuracy values of MMRE less than 25% and PRED25 more than 75% are desirable. However, it is also admitted that estimation results depend heavily on data quality and characteristics.

4.2 Analogy Based Model

The EbA model was produced with the use of BRACE ([7]), a tool that supports the calibration of the method on a given dataset through the use of a genetic algorithm and provides interval estimations for project portfolios using bootstrapping ([8]). Calibration of EbA refers to the empirical

selection of the best combination of distance metric (e.g. Euclidean, Manhattan distance), number of analogies (one or more), statistic for calculation of estimate (mean or median) and size adjustment (yes or no). We calibrated EbA on both ISBSG_SLOC_all_ND and ISBSG_SLOC_C. Precision results for the EbA models are reported on Table 3.

Table 3: Summary results for EbA estimation models.

ISBSG-SLOC-ND		ISBSG-SLOC-C	
MMRE	PRED25	MMRE	PRED25
66,57%	18,83%	49,90%	31,25%

As one can see, both the OLS and the EbA model produce results that are not satisfactory from the accuracy point of view. On the other hand, in both cases residuals were randomly distributed and no bias was observed. Therefore, because such models will be applied on a large number of FLOSS projects, we anticipate that errors will counterbalance each other. As a consequence, although the prediction of a single project effort will not be reliable, we assume that prediction of the overall effort for a large code base will be close to the actual effort needed to implement it.

Also a note on the Multivariate Lindberg-Levy Central Limit Theorem with unequal variances, which makes life easier. Without getting into the detailed mathematics of this, the theorem states that the sums of random variables, regardless of their form, tend to be normally distributed. It also holds that the variables in the sum does not need to come from the same underlying distribution, it basically only requires that only the mean is a mixture of many random variables, none of which is large compared to their sum [9].

4.3 Preliminary Results

Based on these models and the assumptions discussed above, we produced rough estimates of the substitution cost for DEBIAN-1. We assumed that all projects would be substituted / developed using a 3GL type language and that they would be developed from scratch (corresponding to ‘new development’ in ISBSG data). OLS produced only point estimates for the portfolio of DEBIAN-1 projects (no interval estimates have been produced since the tool for their automatic generation is not yet available). These are 437,51 Million Euros for ISBSG-SLOC-ND and 412,68 Million Euros for ISBSG-SLOC-C. The results for EbA are shown in Table 4.

Table 4. Substition costs for DEBIAN-1 5157 software applications (numbers in Million Euros)

	Lower bound	***Most probable***	***Upper bound***
ISBSG-SLOC-ND	264,67	360,55	462,66
ISBSG-SLOC-C	251,30	328,00	426,61

We used a conservative estimate of the salary of a Software Engineer/Developer/Programmer with only one year experience, further taking the average of the salary between Europe and the United States. This amounted to 38.814 Euro per year. We also assumed the average working time per year to be 1600 hours.

5. Validity threats to the study

There is a number of validity threats to our study. In this section, we discuss the most significant of them.

5.1 Use of SLOC as a size metric

SLOC is a physical size metric that has been criticized by many researchers. It is considered that it can not be used to represent accurately effort spent for implementing software because it can not be predicted safely at the beginning of the project, it depends heavily on the programming language used, the programming style of each individual programmer, etc. In our study SLOCs are not predicted, they are just counted at a specific project time slot. Other concerns for SLOCs are still valid. We presume that significant part of the prediction error is due to noise produced by the use of SLOC.

5.2 Use of ISBSG for model building.

ISBSG projects used for calibrating the estimation models may be different than the kind of projects found in Debian. This fact may also be the cause for part of the prediction error.

5.3 Estimation model precision.

Due to various assumptions the precision of the two models is not high. As mentioned above, we consider that only the cumulative effort (total substitution cost) can be used for any useful calculations.

5.4 Debian sample.

The projects used in our study are only a portion of the Debian release considered. Very large projects and very small projects are not taken into account. Such arrangement may have produced a subset of projects biased towards specific software application types and may have affected the kind of projects that participate in our calculation of substitution cost, reducing the generalizability of our results.

6. Conclusions and Future work

We believe that the preliminary rough estimates demonstrate the feasibility of our approach for the calculation of the substitution cost of FLOSS. All point estimates are relatively close to each other and provide a safe order of magnitude for our target cost estimate. In addition, both OLS point estimates fall within EbA intervals. Because ISBSG R 10 data originate from many different companies, scattered around the world, our models are of relatively low precision. For the moment, we base our estimates mainly on physical lines of code, and a couple of project attributes. However as more data describing FLOSS projects become available through the FLOSSMetrics

project or other sources, we intend to build estimation models that will take a variety of project attributes into account. In addition, we will attempt to produce more precise models based on data analysis and the application of other estimation methods (e.g. machine learning approaches). Eventually, our models will be applied to the entire FLOSSMetrics database, providing an informed estimate of the substitution cost of FLOSS by the modern software industry.

Acknowledgments This work is supported by the Free/Libre/Open Source Software Metrics and Benchmarking Study (FLOSSMetrics), Project No: 033982, (http://flossmetrics.org). Sincere thanks to the maintainers of ISBSG, the researchers from Universidad Rey Juan Carlos, in particular Jesus Barahona, Gregorio Robles, Israel Herraiz, and Sandiago Duenas, and Eleni Konstantinou (AUTH PhD student) for their continuous support to our work.

7. References

[1] Boehm, B. 1981 Software Engineering Economics, Prentice Hall.

[2] Mendes, E., Lokan, C. 2008. Replicating studies on cross- vs single-company effort models using the ISBSG database, EMSE, 13, 1 (Feb 2008) 3-37.

[3] Ghosh, R., et al, 2006, FLOSSIMPACT - The impact of Free/Libre/Open Source Software on innovation and competitiveness of the European Union, available online at: http://ec.europa.eu/enterprise/ict/policy/doc/2006-11-20-flossimpact.pdf

[4] Y. Zhou, H. Leung. An empirical analysis of the relationships between structural metrics and unit testability in object-orientedsystems. Technical Report 2008-1, Laboratory for Software Development and Management, http://www.comp.polyu.edu.hk/people/doc/cshleung/, 2008.

[5] Aggarwal, K., Singh, Y., Chandra, P., Puri, M, 2005. Bayesian Regularization in a Neural Network Model to Estimate Lines of Code Using Function Points', Journal of Computer Sciences 1, 4 (2005) 505-509.

[6] Moses, J., Farrow, M., Parrington, N., Smith, P, 2006. A Productivity Benchmarking Case Study using Bayesian credible intervals, Software Quality Journal, 14 (2006) 37-52

[7] Stamelos, I., Angelis, L., Sakellaris, E. 2001. BRACE: BootstRap based Analogy Cost Estimation, in Proceedings of 12th ESCOM (2001) 17-23

[8] Stamelos, I., Angelis, L. Managing Uncertainty in Project Portfolio Cost Estimation, Information & Software Technology, Elsevier, 43, 13 (2001) 7

[9] Greene, W. 2003, Econometric Analysis, 5th ed., Pearson Education.