Determinism and Evolution

Israel Herraiz
Universidad Rey Juan Carlos
Madrid, Spain
herraiz@gsyc.es

Jesus M. Gonzalez-Barahona
Universidad Rey Juan Carlos
Madrid, Spain
jgb@gsyc.es

Gregorio Robles
Universidad Rey Juan Carlos
Madrid, Spain
grex@gsyc.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, we have performed an empirical study on a sample of 3,821 libre (free, open source) software projects, 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.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Reverse engineering, Version control*; D.2.9 [**Software Engineering**]: Management—*Life cycle, Software configuration management, Time estimation*

General Terms

Theory

Keywords

software evolution, time series analysis, self-organized criticality, long term process, short term process

MSR'08, May 10-11, 2008, Leipzig, Germany.

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 [8, 9]. 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 [21]).

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 [26], 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 order to explore if this kind of dynamics is a property of libre software, we have selected a large (3,821) sample of projects, performing an analysis similar to the one by Wu. We have studied the daily time series of changes, focusing on deciding whether their profile were short or long range correlated.

The projects were obtained out of the whole population of projects stored in SourceForge.net, a well known hosting service for libre software projects, that provides a web-based integrated development environment. The data was obtained using the *CVSAnalY SourceForge dataset*[2], maintained by our research group.

[1]In this paper we 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.
[2]http://libresoft.es/Results/CVSAnalY_SF

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 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 [24, 25].

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 [8, 9], 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 we label as the *statistical* and the *physical* approaches. Among the *physical* approaches, as well as the works by Turski already cited, we mention [1, 6, 22]. 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 our 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. We can cite again the works by Godfrey and Tu about the evolution of Linux, or the update of that work by Robles *et al.* [21]. Some studies have tried to use time series analysis (the same approach that we 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 past MSR Challenge [10]). The same approach has been used by other authors [2, 5]. 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 [28, 29, 30].

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.*, we cite [7, 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 [26] 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 recently presented in the IEEE International Conference on Software Maintenance [27].

3. DATA SOURCE

In this study we have considered the evolution of $3,821$ software projects, that are hosted in SourceForge.net (SF.net). This site is a hosting service that offers a web-based integrated development environment for libre software projects.

SF.net is being object of intense study by the FLOSSMole research project [12]. FLOSSMole parses the web pages of SF.net, and creates a database including information regarding all the projects hosted in this site. This information is still mainly metadata (such as the license, number of developers, number of releases, etc) rather than fine granularity information such as change records, source code metrics or defects data.

The database of FLOSSMole has been used by our research group to create a database containing the whole change history of the project. This has been done for all the projects stored in FLOSSMole that fulfill the requirement of having a CVS repository[3]. The last dataset available was obtained in June 2006, and it is publicly available.

That dataset was the result of joining the databases created by the CVSAnalY tool, that was executed directly on the CVS repositories of the projects in SF.net. Among other information, that database contains a register for each commit in the CVS repository for all the projects stored in SF.net (including additional information such as the date of the commit, author, files affected, etc). Therefore, we could easily measure the daily number of changes for each project using the CVSAnalY dataset.

We selected the projects according to the criteria shown in section 4. After the selection process, the sample contained $3,821$ projects.

Table 1 shows some statistical properties of the selected projects. The number of developers is the value stored in the FLOSSMole database. The project page in SF.net contains a field indicating the number of developers that work in the project. That field is increased each time a new developer joins the project. That field is not related to the CVS. Although joining a project is a prerequisite to get access to the CVS, being a developer in the project in SF.net does not imply any participation in the CVS. In other words, that number is an upper bound for the actual number of developers working in the CVS.

Regarding the values labeled as "SF.net age" and "CVS age", those columns indicate the age of the projects in months.

[3]The database is known to contain some errors though, but that affects only to a few projects. We will ignore the influence of those few projects in the global results.

The first value is the number of months that the project has been stored in SF.net. The second value is the difference in months between the dates of the last and first commit in the CVS.

The next two columns (SLOC and number of files) indicate the size of the project in Source Lines of Code (SLOC) and in number of source code files. Basically, SLOC exclude blank and comment lines. Both values, SLOC and number of files, were obtained using the tools SLOCCount[4]. The measured sources were the last checkout (obtained on February 2008) of the CVS repository of each project.

The last column indicates the number of commits that have occurred in the CVS repository.

All the values approximately correspond to June 2006 (the exact dates differ from project to project).

4. METHODOLOGY

The methodology had three main steps:

1. Data retrieval
   We downloaded the *CVSAnalY dataset*, and obtained all the data necessary for the analysis from that database.

2. Time series analysis
   After gathering all the changes history, we calculated the daily number of changes for each project. For each project we obtained a time series that required further processing. We had to apply a smoothing procedure to remove some noise from the data. After removing the noise, we 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, we used regression analysis and then analyzed the distribution of these regressions.

These steps are detailed in the following subsections.

4.1 Data retrieval

Once we downloaded the CVSAnalY dataset, we had to select the sample of projects for this study. There are two main points about the requirements that the project must fulfill: we need active projects, and the projects must be old enough to study their evolution.

About the first requirement, we used the criterion suggested by Capiluppi and Michlmayr [4]. These authors propose that projects with very few developers are probably still in an early stage in their history. Their research has shown that the behaviors of projects at different stages of evolution are different. There are three stages in the evolution of projects. The transition from one phase to another is gained when the project achieves to attract more developers. The first phase in the history of the project is called by Capiluppi and Michlmayr the *cathedral phase*. The next phase is a transition phase to the third, that is called the *bazaar phase*. Those labels are used to match the terms used by Raymond in his paper about the different dynamics of propietary and libre software [20].

Summarizing, and using the terminology of the mentioned paper, we wanted to select all those projects that were likely to be in the bazaar phase. The empirical criterion that we used was to select all the projects with at least 3 developers. We used the value stored in the database of FLOSSMole, that corresponds to the field mentioned in table 1.

[4]Available at http://www.dwheeler.com/sloccount

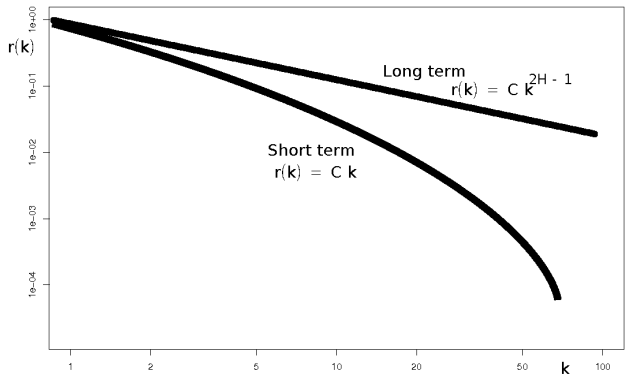

Figure 1: Theoretical profiles of the autocorrelation coefficients of long term and short term processes. This diagram shows the mathematical relationship among the coefficients and the time lags. That equation may be used to obtain the Hurst exponent in the case of long term processes. Note the logarithmic scale of the axis.

The second requirement is having at least one year of history. In the original work by Wu *et al.* where they propose the SOC as the dynamics mechanism for the evolution of libre software, they study a year of changes that was extracted from the whole history of the projects. In order to compare our results with the results of the mentioned work, we decided to study, at least, one year of history of the projects.

We have measured the age of the project in the CVS repository as the difference between the dates of the last and first commits. Table 1 shows the values of age for the sample of projects that have been studied. There are two columns: SF.net age, as explained above, measures the number of months since the project was registered, and CVS age is the value mentioned in this paragraph. Because not all the projects start to use CVS right from the beginning, SF.net age is an upper bound for CVS age.

This selection procedure gave us a set of $3,821$ projects. We obtained the daily number of changes for those projects. We counted only changes made to source code. We identified changes corresponding to source code thanks to the information contained in the LibreSoft's dataset. The tool used, CVSAnalY, is able of identifying the kind of file that was changed in each commit, and marks that change in the database with a determined type (documentation, images, translation files, source code, etc).

Thus, for each project we had a time series containing a point for each day. The value was the number of changes performed on that day.

4.2 Time series analysis

One of the main properties of time series is the autocorrelation coefficients. These are the linear correlation coeffi-

	# Developers	SF.net age	CVS age	SLOC	# Files	# Changes
Min.	3	29	12	10	1	2
Max.	354	92	91	12,028,586	44,174	126,354
Mean	8	64	33	72,903	374	2,417
Median	5	64	29	21,168	142	850
Sd. dev.	12	17	17	341,354	1,165	5,626

Table 1: Statistical properties of the sample of 3821 projects. SF.net age indicates the number of months that the project has existed in SF.net. CVS age indicates the number of months that have elapsed between the first and the last commits in the CVS repository. SLOC measures size in Source Lines of Code (it excludes blank and comment lines).

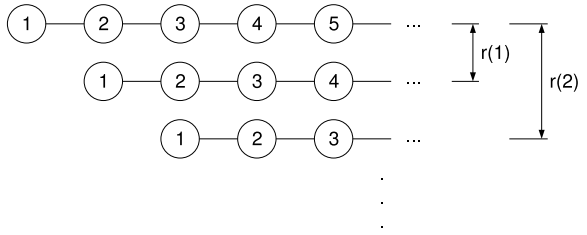

Figure 2: Autocorrelation coefficients in a time series. The coefficients are calculated correlating the series against the same series but shifted one position. If the series has n elements, there are $n-1$ coefficients. In the plot, each circle represents a point in the series. The plot shows how the series is progressively shifted, and how each coefficient is obtaining by linear correlations of the original series and its shiftings.

cient 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 2 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.

Short and long term processes present a very different profile. Figure 1 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 [26]) 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 relationship 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.

A real process would be somewhere among those extreme profiles.

In his PhD thesis [26], Wu gives more details about these same examples of short and long range correlated processes.

In the case of our sample, 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, we applied kernel smoothing [23]. 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. We discuss the influence of the degree of smoothing on the results in the section devoted to the threads to validity. In any case, we have validated this smoothing procedure in other cases [10, 11], and both times 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 our experience [11], 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 [10], we 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

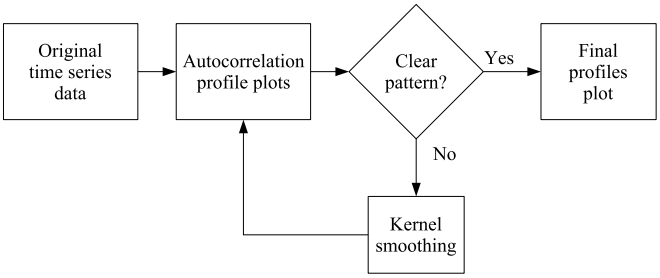

Figure 3: Smoothing and profiles plot process. The original data was very noisy, and smoothing was needed in order to obtain a clear profile.

is, using a different memory parameter for each module), our approach won the challenge and no component had a memory greater than 3 days [10].

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. We describe that step in the next subsection.

4.3 Statistical analysis

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

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

We repeated the mentioned procedure for the $3,821$ projects. The result was a set $3,821$ Pearson coefficients. As shown in table 2, the values of the coefficients ranged from 0.32 to 0.99. 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, we obtained $3,821$ 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 us a set of $3,821$ values. Table 2 summarizes the statistical properties of this set. At a first glance, it seems that the values are distributed around high values (this is for instance $r > 0.8$), which indicates that the processes are more close to a short term process than to a long term one.

Minimum	.3235
Maximum	.9998
Mean	.8429
Median	.8534
Sd. dev.	.1238

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).

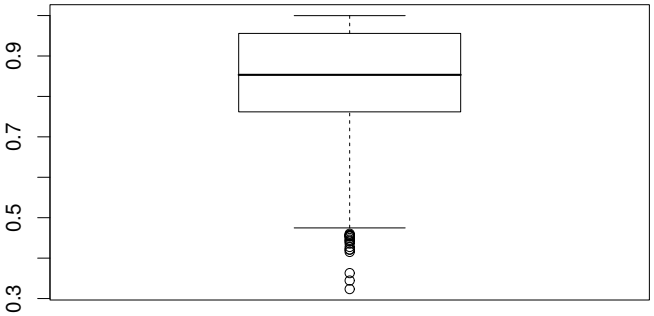

Figure 4: Boxplot of the set of Pearson correlation coefficients. This graphs shows that most of the cases (main box in the plot) are around high values ($\sim$ 0.85) of the coefficient. In other words, most of the projects appear to be short term processes.

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. The results are quite clear. Let us focus on the low values. The boxplot indicates that the values below 0.5 (approximately) may be considered outliers. Most of the values are in the range between 0.75 and 0.95, and the median is around 0.85. Summarizing, most of the projects present a high Pearson coefficient, indicating a strong linear relationship. This would indicate that most of the projects are evolving like short range correlated processes.

An estimation of the density function would help us 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 taking in account the statistical analysis performed on the Pearson coefficients, it seems clear that most of the projects are governed by a short memory dynamics. There are only a few

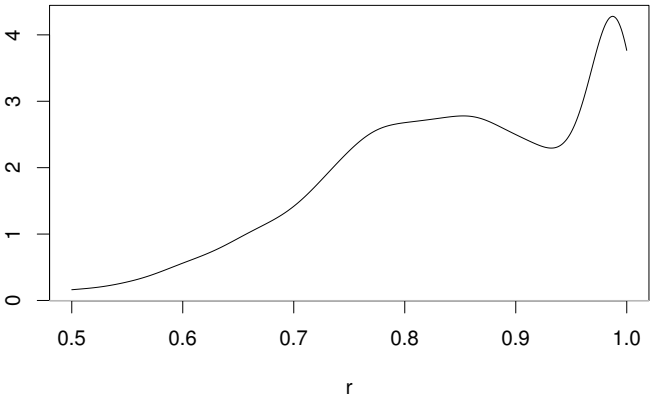

Figure 5: Density function of the set of Pearson correlation coefficients. It seems that there is a group of projects with coefficients very close to 1, and another group simmetrycally distributed around a value of approximately 0.85. Both groups would correspond to short term processes. Long term processes,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

Quantile (%)	r
0	.3235
10	.6739
20	.7394
30	.7807
40	.8178
50	.8534
60	.8906
70	.9312
80	.9783
90	.9932
100	.9998

Table 3: Quantiles of the sample of Pearson correlation coefficients. Less than 40% of the projects present a value lower than 0.8178.

projects that present a profile that would indicate a long memory dynamics.

6. SENSITIVITY ANALYSIS

In the previous section we have shown that most of the Pearson coefficients show high values, meaning that most of the projects under study are evolving like short term 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.

We have considered all the factors that are shown in table 1. We 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. Some of the plots have their vertical axis in logarithmic scale. This is because the kind of distribution of that property. For instance, size 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. In order to make the plots clearer, we have selected the logarithmic vertical axis for those properties. 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 0.7 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 size. In the range of the Pearson coefficient from 0.9 to 1.0, there small, medium and large projects. If we focus in a range of size, we also find projects with a wide range of values of the Pearson coefficient.

This behavior is verified with the rest of properties too. However, the amount of dispersion is different for each property. For instance, the dispersion of the points for the case of CVS age seems to be bigger than the dispersion for the case of size. This is probably because the statistical distributions of those two properties are different. In other words, in the case of size, there are only a few points for very large and very small projects in the plots, but that is because indeed there are only a few projects with those values in the sample.

In any case, the statistical analysis described in section 4 should be repeated using smaller and more coherent groups. The sensitivity analysis shown here is only a first approach to find out if the results are sensible to the different properties of the projects.

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 we 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 at least 80% 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, we have not found a group with that characteristic. The long term evolving projects seem to be a marginality compared to the population of the rest of the projects.

In those results, we 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 process could be a stage in the evolution rather than an dynamical property present in the whole life of the project.

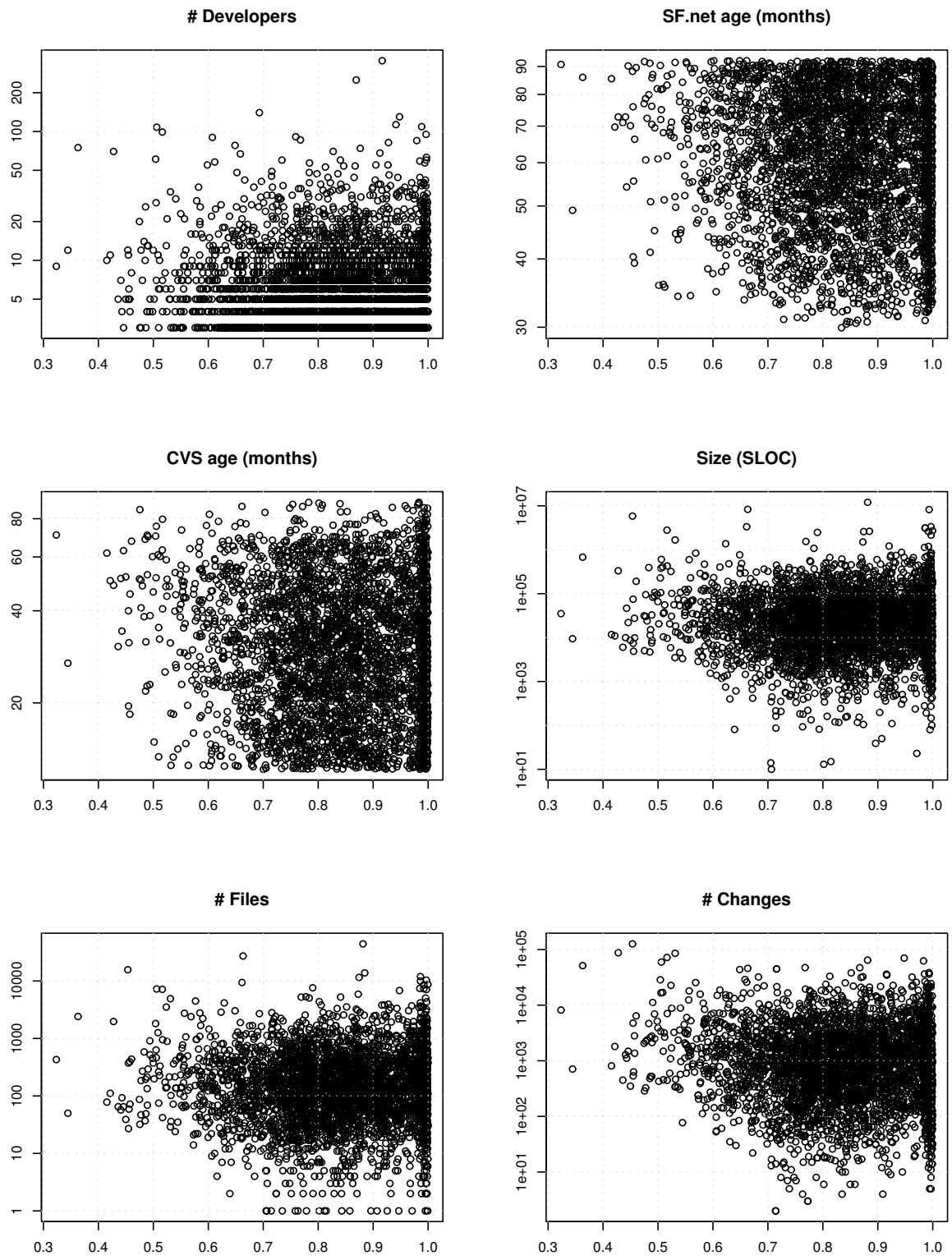

Figure 6: Sensitivity analysis. The plots show the values of each one of the properties shown in table 1 (vertical axis), compared against the Pearson correlation coefficient (horizontal axis). Short term projects present values close to 1. The vertical axis of some of the plots are in logarithmic scale.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

To find out if any of the properties that we have measured for the projects (shown in table 1) influenced the results, we 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. 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, lines of code, or files), or in any age range (both for the above mentioned SF.net and CVS ages).

Therefore, although the projects under study do not form a homogeneous group, the short term evolution dynamics seem to be present in projects with very different properties, suggesting that this dynamics may be an universal property.

8. THREATS TO VALIDITY

We 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. We have done some tests for some isolated projects, and have not observed any modification of the profile (except to make it clearer). We have used the same procedure in previous studies [10, 11], where the results were compared against real data, and in spite of using smoothing, the time series performed well. Therefore we 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. SourceForge.net (SF.net) is known to contain many dead projects, and it has been reported [3] that SF.net may not be a representative source of the libre software development world. In any case, we have studied a large sample of projects, all of them with at least one year of active history, and with a minimum of core developers. Even if those projects have failed, we think that the sample is likely to be representative of typical libre software development (this is, developed in a community, with a large and modular source code base, etc).

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 [26, 27] seems to fit well with the concepts of libre software development. However, in our 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, we have studied the evolution of a large $(3,821)$ sample of cases. Our results show that, at least, 80% of our projects are very short term correlated projects, and probably less than 20% could be considered non-short term processes.

Our study has been made on a heterogeneous set of projects. The differences among the projects might influence the results. For instance, it may happen that small projects present a short memory while large projects present a long memory. We have tested the sensitivity of our statistical analysis, and have found that the results are verified also for smaller and more homogeneous groups. However, the full analysis should be performed on those smaller groups. We plan to do that in a further work.

Furthermore, this study is an example of the possibilities of making large scale empirical research when databases are documented and publicly available. FLOSSMole [12] and our dataset are examples of this kind of collaborative research. In any case, that kind of databases and initiatives should be encouraged. The European Commission-funded project FLOSSMetrics (which has funded in part this work) will provide databases with metrics and facts for thousands of libre software projects. As further work, we plan to repeat the analysis shown in this paper with the projects that FLOSSMetrics will include in its databases. That will help us to find out if the short memory behavior is verified in other projects not stored in SF.net, thus overcoming one of the threats to validity mentioned in the previous section.

Finally, our findings suggest that SOC may not be a good model for a hypothetical theory of software evolution, and that the quest for such a theory must go on.

10. ACKNOWLEDGMENTS

This work has been funded in part by the European Commission, through projects FLOSSMetrics, FP6-IST-5-033982, and Qualipso, FP6-IST-034763, and by the Spanish CICyT, project SobreSalto, TIN2007-66172.

11. 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] K. Beecher, C. Boldyreff, A. Capiluppi, and S. Rank. Evolutionary success of open source software: an investigation into exogenous drivers. In *Third*

International ERCIM Symposium on Software Evolution. ERCIM, 2007.

[4] A. Capiluppi and M. Michlmayr. *Open Source development, adoption and innovation*, chapter From the Cathedral to the Bazaar: An Empirical Study of the Lifecycle of Volunteer Community Projects, pages 31–44. IFIP: International Federation for Information Processing. Springer Boston, 2007.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[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] J. Howison, M. Conklin, and K. Crowston. FLOSSMole: a collaborative repository for FLOSS research data and analyses. *International Journal of Information Technology and Web Engineering*, 1(3):17–26, July-September 2006.

[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] E. S. Raymond. The cathedral and the bazar. *First Monday*, 3(3), March 1998. http://www.firstmonday.dk/issues/issue3_3/raymond/.

[21] 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.

[22] 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.

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

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

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

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

[27] 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.

[28] 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.

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

[30] 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.