On the prediction of the evolution of libre software projects*

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

Daniel M. German
Department of Computer Science
University of Victoria (Canada)
dmg@uvic.ca

Abstract

Libre (free / open source) software development is a complex phenomenon. Many actors (core developers, casual contributors, bug reporters, patch submitters, users, etc.), in many cases volunteers, interact in complex patterns without the constrains of formal hierarchical structures or organizational ties. Understanding this complex behavior with enough detail to build explanatory models suitable for prediction is an open challenge, and few results have been published to date in this area. Therefore statistical, non-explanatory models (such as the traditional regression model) have a clear role, and have been used in some evolution studies. Our proposal goes in this direction, but using a model that we have found more useful: time series analysis. Data available from the source code management repository is used to compute the size of the software over its past life, using this information to estimate the future evolution of the project. In this paper we present this methodology and apply it to three large projects, showing how in these cases predictions are more accurate than regression models, and precise enough to estimate with little error their near future evolutions.

1 Introduction

Libre software[1] projects are usually based on a community of many different actors, ranging from core developers to casual contributors, and even users (who may contribute for instance with bug reports). Most of them behave according to their own interests, in many cases on a volunteer basis. The community formed by those actors shows some structure [8, 22] with different levels of involvement and expertise that may change over time. The management of the project is usually distributed (to some extent), and decisions are difficult to impose, since no formal organizational links or hierarchies are recognized by all the actors. Therefore, developers can not be compelled to do some tasks if they do not want to (even if those tasks are urgent): they may be involved in non fundamental activities, while not contributing to some others with higher priority for the project. There is also usually a lack of predefined requirements, undetailed designs and absence of interprocess documentation [27]. Together with volunteer contributors, some others hired by companies can also be present, in some cases with their own agenda, which usually complicates the picture even more.

In many cases, and despite the lack of apparent plans, these forces and interests result in reliable and mature software which satisfies the needs of many users. Many libre software projects grow continuously over time, and they seem to satisfy most of the requirements of their users. Well-known examples are the Apache web server, the Linux kernel or the Mozilla Firefox web browser.

The problems for forecasting the evolution of the products produced by those projects are clear. If forecasting is already a risky business in traditional software development, where the environment is more constrained, it is even more difficult in these more complex scenarios. However, having a predictive model is undoubtedly a fundamental tool for those interested in the future evolution of those products.

There are several studies proposing models for how certain aspects of libre software products evolve, in some cases including prediction capabilities [3, 10, 26]. However, they fail to provide a comprehensive view of the evolution, probably due to the inherent complexity of the phenomenon and the many different interactions among involved actors. Each individual decision, even when based on self-interest, may seem to the observer as a random event. Therefore, if we focus only on the "low-level" interactions of the community, it is difficult to find meaningful results for the global landscape.

*This work has been funded in part by the European Commission, under the FLOSSMETRICS (FP6-IST-5-033982) and QUALOSS (FP6-IST-5-033547) projects. Israel Herraiz has been funded in part by Consejería de Educación of Comunidad de Madrid and European Social Fund, under grant number 01/FPI/0582/2005.

[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).

On the contrary, if we look at the project from a macroscopic perspective, the overall behavior is not random. Explaining it with a theoretical model may be difficult, but making predictions based on statistical methods may be feasible (as predicting how individual actors will behave in the stock market is difficult, while several statistical methods may be used to predict the quotation of a stock value). In the field of software evolution, the usual statistical models for this kind of prediction are based on regression analysis.

We have followed this approach using a model based on time series analysis (instead of regression). For the purpose of this study we have used the size of the system as the main parameter, but this methodology can be used with other evolution metrics. This proposed methodology has also automated, making the study of a large set of projects and the validation of results straightforward. For the three projects analyzed we found that time series are better are predicting their growth than regression modeling.

Forecasting the near future of a libre software project may be a very useful tool both for the libre software project itself as for those organizations interested on the project. First of all, predicting growth could be used to predict effort, by applying effort estimation methods such as COCOMO[5]. Although we apply in this paper the method to the overall size of the system, it could be also applied to the size of a particular module; for instance, if a company has developers working on a particular module, it could be possible to forecast the effort needed in the following months in that module. Secondly, by estimating other evolution parameters besides size, it would be possible to predict the quality attributes for the project (using a quality model such as QSOS [2], OpenBRR [1], or OSMM [11]).

In the rest of this paper, we first present with more detail the relevant related literature, to later present the methodology we use and a description of the case studies, three well known, large, libre software projects. After that, the results of applying our proposal are presented and discussed, possible threats to validity are considered, and then some conclusions are discussed.

2 Related work

Software evolution has been studied for 30 years now. The seminal work by Lehman [19] stated the *laws of software evolution*, which were revised in a later work [20]. Based on these *laws*, Turski proposed a model to predict the growth of software projects [29], which was later generalized from a discrete model in difference equations to a continuous differential equation model [30].

The model by Turski is based on two assumptions, derived from the *laws* of software evolution:

- The growth of the system is inversely proportional to the system's complexity
- The complexity of the system is proportional to the square of the size of the system (only in the discrete model)

However, these assumptions are not fulfilled in most of the libre software projects. From a first study by Godfrey [15], to the most recent works in the area [24], many libre software projects have been found to be growing with linear or superlinear patterns, in conflict with the first assumption. This questions the validity of these models in the case of some libre software projects.

The idea of using time series analysis to predict software evolution is not new. Already in the period from 1985 to 1988 several papers [31, 32, 33] used statistical methods, including time series analysis, to model software evolution. For instance, in [33] time series ARIMA models are used to predict the evolution in the maintenance phase of a software project, using sampling periods of about one month.

Later, Kemerer and Slaughter [17] followed this line of research proposing an ARIMA model which is able to predict the monthly number of changes of a software project. As in the previous cases, Kemerer and Slaughter did not obtain very good results applying time series, because the phenomenon they were studying (number of changes) was close to a random process.

Other authors have applied ARIMA models and time series methods specifically to predict the evolution of a software project. In [6], the authors applied time series analysis to predict the evolution of the Linux kernel. They took a small subset of all the releases of Linux, and tried to predict the size of future releases. The results were mostly satisfactory, but for some releases the model gave results with large relative errors. In [4], the authors applied the same methods to model the evolution of software clones in a libre software project.

In 2002, Fuentetaja and Bagert [13] also explored the use of time series to obtain a model for the evolution of software projects. However they did not provide a model to predict the evolution but some tools which could be useful to obtain such a model.

More recently, Dalle *et al.* [9] have applied signal processing techniques to gather information from the time series obtained from versioning systems of libre software projects. They were inspired by a previous work [16], which applied time series analysis concepts to visualize historical information from software projects.

However, time series methods do not seem to be the most popular technique used by the research community for the analysis of software evolution. Studies in this field prefer statistical regression techniques, as is the case of [12, 15, 18, 24]. Regression models are convenient for extracting the main trend from the evolution curves, but they fail to predict the short term behavior of the project. We will show in this paper that time series analysis are more suitable than

regression models for software evolution analysis. Principal component analysis has also been applied, in combination with regression techniques in [23]. In that paper, a model for the evolution of some operating systems (both libre and non-libre software) is obtained.

There also some other non-statistical models proposed to understand the evolution and behavior of the projects. Robles *et al.* [26] propose a model based on the *stigmergy* concept, which assumes that communication between individuals happens through stimuli caused by changes in the environment, and not by directly exchanging information. This model was applied to find out how developers join projects, and how the work by these developers affect the overall evolution of the project. Another model was proposed in [3], which tried to describe all the processes found in a libre software project by means of differential equations. The output of the model is the future values of some parameters of the projects (among them, size). The authors did not validate the results obtained with this model though.

3 Methodology

The main objective of this methodology is to study the evolution in size of the products delivered by a libre software development community over its history. As sampling period we have chosen one day. For that task, we first need to obtain the source code of the project for every day over time. Instead of retrieving and measuring all the code at every day (which leads to unnecessary measurement of those files that have not changed since the last day), only files that have changed during that period are retrieved and measured. Once they are measured this way, results are aggregated for each day to get the corresponding size for the whole project.

In order to measure size, the selected metric is Source Lines of Code (SLOC), as defined in [7]:

> 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.

To count SLOCs we use the SLOCCount tool[2]. We decided to obtain this metric because it has been traditionally used in the study of the evolution of libre software projects (as an example, we cite [15, 24]).

3.1 Data collection

More in detail, we start by getting a copy of the CVS repository of the project to be studied. Usually, researchers have to query the original CVS repository of the project, overloading the server and making the retrieval slow. Sometimes in this process, the project sets a ban as they interpret that the repository is used for a different purpose than the original one. In our case, all case studies offer the possibility of obtaining their CVS repository using the *CVSup* or *rsync* protocols. Therefore we can replicate the original server to a local machine, and use it to perform our analysis. This avoids overloading the project's server, while it allows us to repeat our study as many times as we want. In that sense, the selected case studies are *research friendly*.

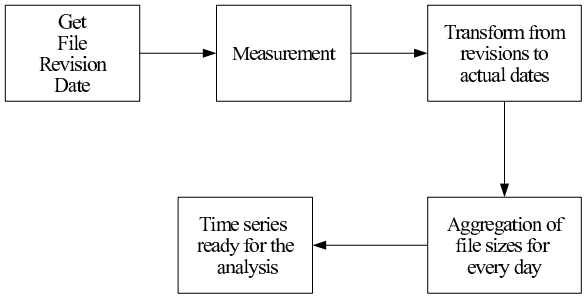

Figure 1. Steps followed to collect the data

Once we had set up our local CVS server using the copy of the repository, we used the softChange tool [14] to analyze it. The steps followed to collect the data are summarized in figure 1. We used the database generated by softChange to obtain the information needed for every revision in the CVS. A revision in the CVS can be understood as a point of change. Every time a change is committed to the CVS, a new revision is created. The database obtained using softChange contained a table with all the information needed to identify the revision. In particular, from this database we read the revision identification tag, the filename, a flag to find out whether or not the revision is in the main trunk, a flag to find out whether or not the file was removed in that revision, the date and the time for the revision. With the filename and the revision identification tag we can obtain the original file from the CVS. With the help of the flags, we can retrieve only files in the main trunk (avoiding the files in parallel branches) and avoid revisions where the files were deleted. And finally, the date and the time allows us to sort all revisions in a chronological order. Once we obtained these data, we proceeded to download all revisions, and to measure them. We stored the results in a new table in the database. Besides files deleted and not in the main trunk, we ignored also those not written in the C programming language, including header files which have not been considered.

After this step, we had a table with all the revisions in the CVS, and their sizes. To reconstruct the evolution of the projects over natural time, from the table generated in

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

the last step, we took the revisions for every day. If a file had more than one revision in the given day, we obtained only the last one (given by the hour field). We built a tree of sources, initially empty. Then, we applied the same changes to our tree than those that were made to the real source code tree. Every leaf in our tree was not a file, but metadata containing the size of the file and the information needed to identify the file (revision id and file path). Therefore, for every day, we had a tree containing the sources of the project as they were in the considered day (that is why we ignored deleted revisions, files that are deleted are not present anymore in the sources tree). Hence, instead of the actual files we had the values of their size. For every day we aggregated the values of all files; i.e., the overall size was the addition of the sizes of all files. By means of this procedure we obtained the results for the whole source code tree on a daily manner since the beginning of the project (see table 2).

3.2 Analysis

After the data collection step, we have a list of values with the size of the case studies, in SLOCs, and a point for every day since the beginning of the project until the last considered date. This is, we have collected a time series of the size for each case study.

These series can be predicted based on past values. In order to achieve such prediction, the data needs to be internally correlated. Internal autocorrelation means that a value in the future depends on some values in the past. Therefore, a model based on past values can be obtained to forecast the future. If there is no autocorrelation, this model can not be obtained. The degree of internal autocorrelation in the data is measured by the autocorrelation coefficients and the partial autocorrelation coefficients.

To obtain the prediction of future values, we apply an ARIMA model to each one of the cases. The steps followed for the analysis of the series are summarized in figure 2. ARIMA models are linear combinations of past values of the series, weighted by some coefficients. To obtain those coefficients we need to identify three different parameters: p, d and q. A seasonal component can be added to the model. For instance, if we would know that any of the projects is releasing a new version with a stable period (for instance, a new release each six months), we could aggregate that information to the model, in order to increase the goodness of the prediction. However, we have not tried to add a seasonal component to the models in the present study.

The values for the parameters are obtained applying the Box-Jenkins method [21]. The first requirement to apply this method is that the data needs to be stationary. The number of differences that must be applied to the data is the value of the parameter d. In our case, as we will show in the Results section, all the series were close to linear, meaning that stationary data can be obtained applying the first difference. In other words, d was 1 for all the case studies.

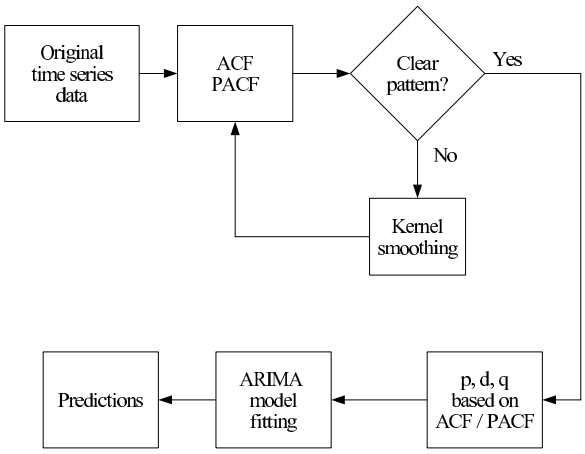

Figure 2. Steps followed in the time series analysis of the data

	$q = 0$	$p = 0$	$p \neq 0$ $q \neq 0$
ACF	Tails off	q significant coefficients	Tails off
PACF	p significant coefficients	Tails off	Tails off

Table 1. Criteria to select the values of p and q in an ARIMA model. ACF means autocorrelation function. PACF means partial autocorrelation function. Reproduced from [28].

In order to obtain the values of p and q, we must inspect the plot of the autocorrelation and partial autocorrelation coefficients. The criteria to choose the values of p and q are explained in table 1. To apply those criteria, we have to plot both the autocorrelation coefficients function, and the partial autocorrelation functions (these functions are described in [21]).

However, if the data is noisy, the autocorrelation and partial autocorrelation functions will not show a clear pattern, and it will not be possible to apply the criteria shown in table 1. In order to remove the noise of our samples, we have applied *kernel smoothing*, as described in [28]. This *kernel smoothing* filter makes the series smoother by introducing some autocorrelation in the data. In the Results section, we will discuss the influence of the degree of smoothing in the validity of the model.

After the filtering, the autocorrelation and partial autocorrelation functions will show a clear pattern. For instance, see figure 6. In order to find out whether a coefficient is

significant or not, we consider that the coefficients are normally distributed, and discard those coefficients greater or lower than two times the standard deviation. This is the requirement generally accepted, and recommended by the literature [21, 28]. In the case of figure 6, all coefficients outside the region limited by dashed lines fulfill this requirement. Therefore, the autocorrelation coefficients function slowly tails off, and the partial autocorrelation coefficients drops to zero after 9 coefficients (in other words, there are 9 significant coefficients). Following the criteria of table 1, this would mean $p = 9$ and $q = 0$. After choosing the values of the parameters, the model can be fitted, and so we are able to obtain the predicted values. The Results section discusses the p and q and obtained values for all case studies.

4 Case studies

The described methodology has been applied to three large, long-lived libre software projects. All of them are older than ten years, therefore providing enough history to study the evolution of the system. The selected projects are described with more detail in the following subsections.

Table 2 contains a summary of the case studies, and the period of time studied. We have studied the whole history available in CVS for each of the projects. In all cases, we have more than 10 years of history.

4.1 FreeBSD

FreeBSD is a Unix-like operating system descended from AT&T UNIX via the Berkeley Software Distribution (BSD) branch through the 386BSD and 4.4BSD operating systems. The development of FreeBSD is lead by a community of volunteers. FreeBSD is released under the BSD license, which is considered a libre software license.

FreeBSD is developed as a complete operating system. The kernel, device drivers and all of the *userland* utilities, such as the shell, are held in the same source code revision tracking tree (CVS). This is in contrast to GNU/Linux, a similar but better-known operating system, in which the kernel is developed by one set of developers; *userland* utilities and applications by others, such as the GNU project; and all are packaged together by other groups and published as Linux distributions.

We studied only the kernel of FreeBSD, included in the `src/sys` module in the CVS repository. We ignored everything not part of the main development trunk.

Some large jumps were observed in the growth plots. After careful inspection of the revisions in the CVS, we could explain all jumps as part of the development and maintenance process of FreeBSD. For example, May 29th 2001 the module (sys/contrib/dev/acpica/Subsystem) was removed, because it was old code for ACPI support. This module was replaced by sys/dev/acpica on May 28th 2000, causing a positive jump. Another example occurred in December 12th 2005, when read-only support for the XFS filesystem was added to the kernel. As all files involved in these jumps were part of the development process (they were written by human developers and not automatically generated), we decided not to remove them from our study.

4.2 NetBSD

NetBSD is a version of the Unix-like BSD computer operating system. It was the second libre software BSD variant to be formally released, after 386BSD, and continues to be actively developed. Noted for its portability and quality of design and implementation, it is often used in embedded systems and as a starting point for the porting of other operating systems to new architectures. The development of NetBSD is lead by a community of volunteers. NetBSD is also released under the BSD license, and therefore libre software.

When plotting the values of the metrics over time we could not observe any jump, so no further investigation was required.

4.3 PostgreSQL

PostgreSQL is an object-relational database management system. The development of PostgreSQL has not been lead by a single company, but is mainly driven by a community of developers and users, including some companies backing it and offering services and support around the software. The origins of PostgreSQL can be tracked to the Ingres project started in the 1970s at the University of California at Berkeley, but since its beginnings in the mid-80s it has evolved on its own way. Although the design and conception is clearly influenced by Ingres, not much code (if any) from it persists in the current version.

From the whole source tree available in the CVS repository, we have studied only the code contained in the `src` module. We have also ignored all code not included in the main development trunk.

At first, we observed some strange large jumps (both, positive and negative). After a careful look at the commit logs and at the mailing lists, we found the reasons for these gaps. First of all, two automatically generated C files had been added to the CVS repository. Some time later, they were removed. And again, they were added some months later. As these files were not been written by developers directly (they are the output of using YACC, a tool which automatically generates C code), we decided to remove them from the sample. The affected files were src/interfaces/ecpg/preproc/preproc.c (2758 SLOC in its

Project	First date	Last date	Months	SLOC	Num. of files
FreeBSD	1993-03-24	2006-06-05	161	1414641	2703
NetBSD	1993-03-21	2006-06-07	161	1999626	6243
PostgreSQL	1996-07-09	2006-07-18	122	289864	593

Table 2. Summary of the case studies. Size is shown both in SLOCs and number of source code files (excluding header files) in the last date considered.

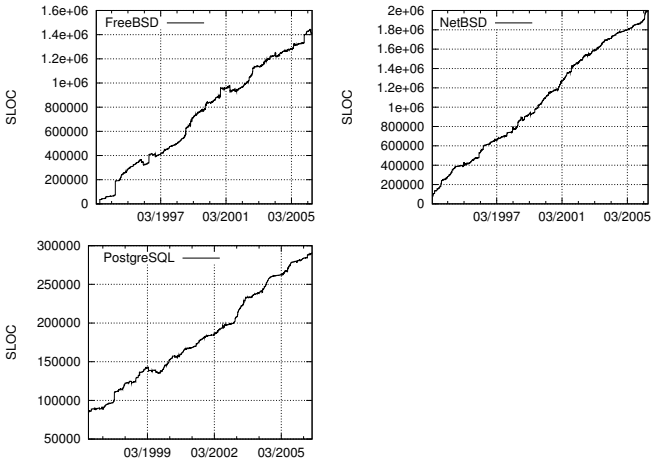

Figure 3. Plots of SLOC over time for all case studies, according to the restrictions discussed in the Case Studies section.

last version) and src/interfaces/ecpg/preproc/pgc.c (16473 SLOC in its last version).

Another source of noise in our curves was the module for ODBC support in PostgreSQL. The development group for ODBC was different that for the main trunk of PostgreSQL. However, all the code was included in the same repository and module. At some point the ODBC module was "branched" into an independent repository and was removed from the main trunk of the CVS, causing a big negative gap in the figures. For this reason we determined that it was reasonable to remove from our study any file that belongs to it (src/interfaces/odbc/*).

5 Results

In this section we describe the results obtained. We show how we obtained the ARIMA models in order to forecast the near future of the project, by applying the Box-Jenkins method.

The first obvious graph is the evolution of the metric over time, shown in figure 3 for all case studies.

The growth of all projects is very close to linear. When correlating SLOC against number of days of lifetime, the correlation coefficients are $R^2 = 0.9899$ for FreeBSD, $R^2 = 0.9945$ for NetBSD and $R^2 = 0.9888$ for PostgreSQL. It seems that the linear regression models are quite good. They are indeed; however, as we show at the end of this section, even with those good correlation coefficients, the time series model performs better than the regression model when predicting the last year of history of the three case studies.

Let us try now to obtain an ARIMA model for each one of the case studies. As described in the Methodology section, we need to identify the values for the three parameters of the model: p, d and q. As the good linear regression coefficients show, the series are close to be linear. Therefore, we can select $d = 1$ in order to get stationary data. To select p and q we need to plot the autocorrelation and partial autocorrelation functions. If we try to do so with the original data (this is, without any filtering), we do not obtain a clear pattern in order to apply the criteria described in the Methodology section (summarized in table 1). For instance, figure 4 shows the autocorrelation and partial autocorrelation coefficients for the first difference of the FreeBSD series; the coefficients are shown for the first ten lags, this is, it shows the internal autocorrelation among the last ten points of the series. The number of lags is not important, as long as enough lags are shown in order to find out the exact pattern of the plots. In the case of the autocorrelation coefficients function, only the first coefficient is significant, and after that lag, all coefficients suddenly drop to zero. In the case of the partial autocorrelation coefficients, all coefficients are very low, and we should discard all of them attending to the criteria described in the Methodology section. What we are seeing in figure 4 is that the noise is reducing the internal autocorrelation of the series, and so any of the coefficients is significant.

We need therefore to filter the original series in order to remove the noise. Figure 5 shows the first difference of the FreeBSD series before and after applying the filtering process. The plot in the top of that figure is the original data, the plot in the middle is the filtered series, applying a *kernel smoothing* with a bandwidth of 10. The plot in the bottom is the filtered series, with the same smoothing with a bandwidth of 50. Smoothing removes some noise from the series, and the trend in the data appears now much clearer. We applied smoothing with a bandwidth of 50 to the series

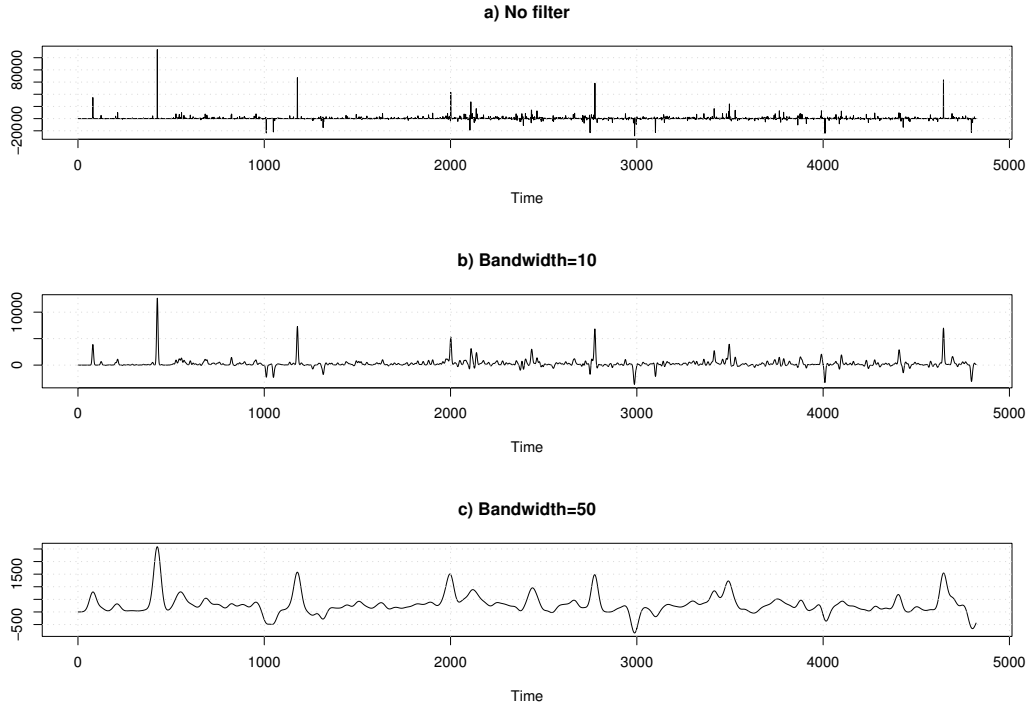

Figure 5. Original series and filtered series for FreeBSD

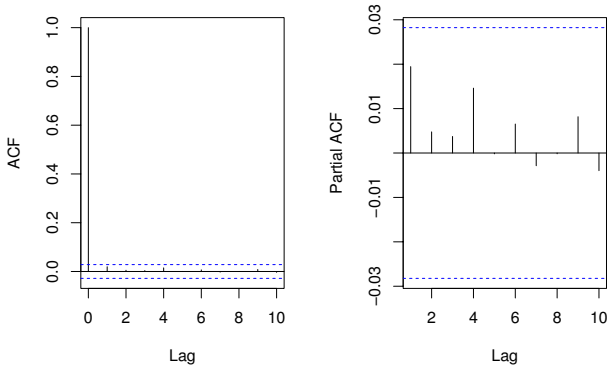

Figure 4. Autocorrelation (left) and partial autocorrelation (right) coefficients for the first ten lags of the difference of the FreeBSD series (no filter applied).

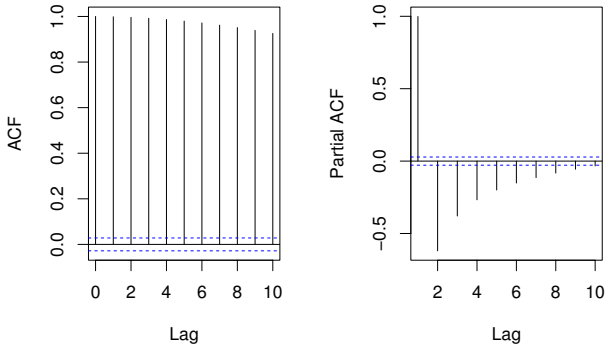

Figure 6. Autocorrelation (left) and partial autocorrelation (right) coefficients for the first ten lags of the filtered FreeBSD series (first difference).

of the three case studies.

After the filtering process, the pattern in the autocorrelation and partial autocorrelation functions is much clearer. Figure 6 shows those plots. According to table 1, we would choose $p = 9$ and $q = 0$, because the autocorrelation function slowly tails off to zero, and the partial autocorrelation function drops to zero after 9 lags.

In the case of NetBSD, the pattern was the same with 6 significant lags in the partial autocorrelation coefficients. Therefore $q = 0$ and $p = 6$. In the case of PostgreSQL, the pattern was the same with $q = 0$ and $p = 7$.

As we have said, to test the goodness of the obtained models, we divided our series in two sets: the training set and the test set. The training set was built with all series but the last year. The last year was the test set[3]. The values obtained from the model were the first difference of the

[3]The test sets contained 321 days for FreeBSD, 326 days for NetBSD and 361 days for PostgreSQL. The number of days is almost one year for all cases.

Case study	MSRE	Sd. dev.
FreeBSD	3.93	3.28
NetBSD	1.80	1.28
PostgreSQL	1.48	1.86

Table 3. Mean squared relative errors (MSRE) and standard deviation of the squared relative errors (Sd. dev.) for the time series models.

Case study	MSRE	Sd. dev.
FreeBSD	16.89	14.82
NetBSD	15.94	8.65
PostgreSQL	6.86	4.75

Table 4. Mean squared relative errors (MSRE) and standard deviation of the squared relative errors (Sd. dev.) for the regression models.

forecasted series. We made the inverse operation of the first difference, and added the value of the actual series at the beginning of the test interval. Then we compared these forecasted values with the actual values.

After fitting the time series model using the training set, we predicted the next year, and compared the forecasted values against the actual values contained in the test set. Table 3 contains the mean squared relative errors for the three case studies.

We repeated the same procedure with a linear model obtained by statistical regression using least squares. We correlated SLOC against number of the days of lifetime, for all the data but the last year (this is, we used the training set for least squares regression). Then we compared the last year forecasted with the regression model against the actual values (the test set). For the case of PostgreSQL the correlation coefficient with the training set was $R^2 = 0.9890$ for FreeBSD, $R^2 = 0.9946$ for NetBSD and $R^2 = 0.9856$ for PostgreSQL. Table 4 shows the mean squared relative errors for the three case studies.

When comparing table 4 to table 3, we can easily see that the relative errors are greater using linear regression models than time series models.

Therefore the time series based ARIMA models can predict the growth in the next year of a project with lower error than regression models. We want to remark the high correlation coefficients obtained with linear models. Even with those quite good correlation coefficients, the ARIMA models perform better than the regression models.

6 Threats to validity

6.1 Limited number of case studies

This study has been performed using only three case studies. The three case studies are quite similar among them: long lived projects, large software systems. Two of them are operating system kernels with the same historical origin (BSD); the other case study is a database management system.

The results may vary if there is not enough history in order to fit the models, this is, if the projects are not long lived. Although the internal memory of the models presented in this papers is low (about one week, as we discuss in the next subsection), the fitting process (this is, to obtain the values of the coefficients of the model based on the past data and the values of the parameters p, d and q) may throw different results for these coefficients if the size of the set used to fit the model is smaller. The size of the system is another important factor: the issues that arise in a large project are not the same than in a small project. Moreover, we have selected a metric (size measured in SLOCs) which is the result of the aggregation of many different stochastic processes. This aggregation of stochastic processes may be not as easily predicted if the size of the system were smaller, because the uncertainty would be greater.

We think that this methodology could be also applied to other large, long lived, libre software projects, but we have not tested the results with other case studies. In order to predict size, the data collection process and the level of granularity do not affect the results of the model in our opinion. This could be different with other metrics or parameters though.

On the other hand, if the projects are not long lived, and their size is not large, and if the sort of studied files is also different, the performance of a time series model may greatly vary.

6.2 Filtering and fitting

The filtering process may affect also the performance of the time series models. Although we have compared the predictions against actual values (this is, non filtered), the filtering process adds internal autocorrelation to the series. If the autocorrelation added to the series is greater than necessary, the result would be probably an overfitted model. However, we do not think that is the case in this study. The parameters of the models are related to the internal memory of the model. As the period between measurements is one day, and the parameters were around 6, 7, the internal memory of the model is about one week (this is, the value today is directly related to the values in the last week). A periodicity of one week fits quite well with the phenomenon

that we are studying (software development and evolution). Another evidence of the suitability of the model is the comparison against actual values: the test set. That set of values was not used at all to obtain and fit the models.

The selection of the training and test sets will make the mean squared relative error values change. However, we have tried different sets, and in all of them the time series models performed better than the regression models. Therefore, we do not think that our conclusions are affected by the selection of the training and test sets.

6.3 Data collection

The data collection process may be also a threat to validity. We have considered only C files, excluding header files. The results may vary if we also consider other sort of files, or if we take all of them (this is, documentation and other files as well) as they show other behaviors [25]. We have removed some files from the sample of one of the case studies, because those files were not written by human developers. In other words, the statistical distribution of the size of the files can be different if we change the sort of files studied, or if we consider also files that are automatically generated. This difference in the statistical distribution of the size across the whole sources tree may affect the suitability of the time series methods in order to forecast the near future of the projects.

7 Further work

ARIMA models may include also a seasonal component, in order to increase the forecasting performance. Some libre software projects are known to release new versions with a fixed schedule[4]. This information could be added to the models. In the case of our case studies, the time between releases is not constant. However, as figure 5 seems to show, there is a seasonal pattern in the data. We plan to find out if that pattern is related to any event in the projects (for instance, a new release of the project), and incorporate that information to the ARIMA models.

The growth curves for the three case studies present a linear profile. ARIMA models can predict any time series, not necessarily linear. Actually, as we are comparing against regression models, linearity would be a factor which would help regression models to perform better. In any case, we should test the goodness of time series models against regression models with other sort of profiles (such as superlinear, which is found in some well known large libre software projects).

We intend to compare also this methodology against other predictive methods. As an example, we cite [23] that studied the evolution of operating systems using principal component analysis in combination with regression techniques.

We also intend to apply this methodology to other parameters of the project, in order to test the performance of the model for effort prediction, to predict the quality attributes of the project by means of a quality model such as QSOS, OpenBRR, or OSMM.

[4]For instance, the GNOME project releases a new version every 6 months.

8 Conclusions

In this paper we have presented a methodology for predicting the near future of the size a software project based on time series analysis. It uses information available in source code management repositories, and can be fully automated, performing an analysis based on an optimal number of measurements (which allows for sampling periods as small as one day).

This methodology has been used to measure and predict the size of three large libre software projects over their whole history (more than 10 years), with a sampling period of 1 day. The data was obtained from publicly available sources (the CVS repositories of the projects).

The Box-Jenkins method was applied to obtain a predictive model, which implied filtering the series. These filtered series were the input to our ARIMA models. To validate them, we used as input all data except those corresponding to the last year. We then used the actual values found for the last year to compare with the results of the prediction models. We repeated the same procedure applying linear regression by least squares. The linear models, in spite of the high correlation coefficients, performed worse than the ARIMA models.

Therefore, we consider time series analysis as a good candidate to estimate the future evolution of libre software products, better than regression models, at least for our case studies. Although we applied our methodology to the time series of SLOC, this methodology can be applied to predict the evolution of any other parameter of the project.

References

[1] Business readiness rating for open source. a proposed open standard to facilitate assessment and adoption of open source software. Technical Report BRR 2005 - RFC 1, OpenBRR, 2005.

[2] Method for qualication and selection of open source software (qsos). Technical Report version 1.6, Atos Origin, April 2006.

[3] 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.

[4] 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.

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

[6] 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.

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

[8] K. Crowston and J. Howison. The social structure of free and open source software development. *First Monday*, 10(2), February 2005. http://www.firstmonday.dk/issues/issue10_2/crowston/.

[9] J.-M. Dalle, L. Daudet, and M. den Besten. Mining cvs signals. In *Proceedings of the Workshop on Public Data about Software Development*, pages 12–21, Como, Italy, 2006.

[10] 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.

[11] F.-W. Duijnhouwer and C. Widdows. Open source maturity model. Technical Report 1.5.3, Capgemini, August 2003.

[12] 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.

[13] E. Fuentetaja and D. J. Bagert. Software Evolution from a Time-Series perspective. In *Proceedings of the International Conference on Software Maintenance*, pages 226–229, 2002.

[14] D. M. German and A. Hindle. Visualizing the evolution of software using softChange. *International Journal of Software Engineering and Knowledge Engineering*, 16(1):5–21, 2006.

[15] 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.

[16] A. E. Hassan, J. Wu, and R. C. Holt. Visualizing historical data using spectrographs. In *Proceedings of the International Software Metrics Symposium*, Como, Italy, 2005.

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

[18] 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.

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

[20] 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.

[21] S. G. Makridakis, S. C. Wheelwright, and R. J. Hyndman. *Forecasting: Methods and Applications*. John Wiley & Sons, Ltd., January 1998.

[22] 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.

[23] 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.

[24] 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.

[25] G. Robles, J. M. Gonzlez-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*, 79(9):1233–1248, September 2006.

[26] 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.

[27] W. Scacchi. Free and Open Source development practices in the game community. *IEEE Software*, 21(1):59–66, 2004.

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

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

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

[31] 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.

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

[33] 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.