Remote analysis and measurement of libre software systems by means of the CVSAnalY tool

Gregorio Robles
Universidad Rey Juan Carlos
grex@gsyc.escet.urjc.es

Stefan Koch
Wirtschaftsuniversität Wien
stefan.koch@wu-wien.ac.at

Jesús M. González-Barahona
Universidad Rey Juan Carlos
jgb@gsyc.escet.urjc.es

Abstract

Libre (free, open source) software is one of the paradigmatic cases where heavy use of telematic tools and user-driven software development are key points. This paper proposes a methodology for measuring and analyzing remotely big libre software projects using publicly-available data from their version control repositories. By means of a tool called CVSAnalY that has been implemented following this methodology, measurements and analyses can be made in an automatic and non-intrusive way, providing real-time and historical data about the project and its contributors.

Keywords: *Mining source code repositories, empirical software engineering, libre software engineering*

1 Introduction

The way software is produced has changed radically in the last two decades. Among other circumstances, the arise of the Internet has brought a change in software production paradigms and an increase in the number of end users. That way, it is not uncommon that software development is done with a distributed team and that fast user feedback is possible, both things by means of telematic tools. Also, versions of the software are released often in order to gain momentum from user's feedback.

One of the software production fields where all the aforementioned characteristics are given is the libre (free/open source) software world. In that area there exists a big synergy between developers and users to the point that sometimes it is often not possible to distinguish these groups. In fact, many (if not all) developers are really power-users that have the required programming capabilities to find a solution to their software needs [15]. Hence, the study of libre software can be seen as a paradigmatic case of a production environment where users have a big implication, even leading the development.

Due to the distributed nature of this development paradigm, cooperation and communication needs to be achieved using telematic tools. Data from these sources allows for an almost-automatic remote analysis and measurement of both software product and underlying processes. In fact, remote analysis is necessary, as a central authority or location like office is absent, preventing on-site evaluations. In addition, all interested parties including users and developers are distributed, and therefore need to perform and also access all measurements and analyses remotely.

Libre software projects go from very small ones with one developer commited to his program to large-scale global projects where thousands of developers interact [10]. Especially most of the bigger projects follow a way of organization that has been called the "bazaar"-style development [15] whose aim is to be as near to the end user as possible, giving users even a co-developer status. In [14] it is shown that such big libre software projects are composed mostly of 10 to 15 core developers who lead the software process, a group of around one order of magnitude larger that participate in minor development tasks (bug fixes, etc.) and a final group around another order of magnitude that helps by other means (bug reports, etc.).

Several research groups have focused their attention to the libre software phenomenon in the last years, so that several views of this paradigm can be found. For instance, [7] offers a software evolution analysis of the Linux kernel - without doubt the most known libre software project- following the classical software evolution point of view [13]. Others have paid attention to economic parameters [12] and have investigated how well classical software cost prediction models as among others [2] can be applied.

This paper presents an empirical analysis of libre software projects that can be made automatically, non-intrusively and remotely from public-available data. The source of the data that is measured and afterwards analyzed is taken from the source versioning systems that most libre software projects use, the CVS (Concurrent Versions System). The CVS contains the current state of the source code as well as all the previous versions of the code. It serves as a basis for developer interaction and group work.

Further possible publicly available data sources would

include mailing lists and bug tracking systems. All have already been used for quantitative research on free/open source software projects. Koch and Schneider have used the mailing lists of the GNOME project [12] to further characterise developers and overall project growth showing that a significant number of postings is made by the relatively small sub-group of programmers, and Mockus et al. have used the bug tracking archives [14]. Especially the last point is problematic for the automated methodology applied here, as there is no single bug tracking system used by nearly all libre software projects (like CVS for version-control), but several competing implementations like Bugzilla or GNU GNATS.

The structure of the paper is as follows: in the next section the methodology and inner functioning of the tool that has been built to measure and analyze CVS repositories will be presented. In the following three sections, results offered by this tool for a real-world large libre software project will be given; first general results of the whole repository, then measures given by modules and finally measures realted to developers are exposed. The last section gives in short the conclusions that we have found after using this tool and possible future research lines.

2 CVSAnalY: analyzing CVS repositories

The methodology of CVSAnalY is based on the analysis of the CVS log entries, although other methodologies have been also proposed in order to automatise the analysis of libre software products, as it can be seen in [17] and [5]. In CVSAnalY any interaction -also called commit- a commiter[1] does with the central versioning system repository is logged with following data associated: commiter name, date, file, revision number, lines added, lines removed and an explanatory comment introduced by the commiter. There is some file-specific information that can also be extracted as for instance if the file has been removed[2]. On the other hand, the human-inserted comment can also be parsed in order to see if the commit corresponds to an external contribution or even to an automated script.

Basically CVSAnalY consists of three steps: preprocessing, database insertion and postprocessing.

The preprocessing includes downloading the sources from the CVS repository of the project in study. Afterwards, aggregated modules [3] have to be removed to avoid counting commits several times. Once this is done, the logs are retrieved and parsed to transform the information contained in log format into a more structured format (SQL for databases or XML for data exchange).

Besides the information for every commit there is other data that is obtained from the parsing that requires some attention. Although username changes occur seldom, some entries for commiters have been merged due to a change. For instance, in the KDE project commiters usually get a CVS account prior to an organization email address. If they afterwards are assigned an email address the username of email and CVS have to be identical for purposes of a clearer organization. If the username in the email address is different from the CVS username, the latter is synced with the former one and for our accounting both usernames have to be merged.

While being parsed each file is also matched for its type. Usually this is done by looking at its extension, although other common filenames (for instance README or TODO) are also looked for. The goal of this separation is to identify different contributor groups that work on the software, so besides source code files the following filetypes are also considered: documentation (including web pages), images, translation (generally internationalization and localization), user interface and sound files. Files that don't match any extension or particular filename are accounted as unknown.

CVS also has some peculiarities when introducing contents for the first time (this action is called check-in). The initial version (with version number 1.1.1.1) is not considered in our computation as it is the same as the second one (which has version number 1.1). The number of aggregated and removed lines in CVS are computed from this initial version. This means that the first commit (the initial check-in) logs as if 0 lines were added. This does not correspond to reality. In order to obtain the actual number of LOCs in the first version we count the LOCs by means of the UNIX wc tool of the latest version, subtracting the added lines and adding the removed lines.

The comment attached to the commit is usually forwarded to a mailing list so that developers keep track of the latest changes in CVS. Some projects have also conventions so that certain commits do not produce a message to the mailing list as it is supposed that the action they have performed does not require any notification. A good example of the pertinent use of "silent" commits comes from the existence of bots that do several tasks automatically. In any case, such conventions are not limited to non-human bots, as human commiters may also use them. In a large community -as it is the case for the ones we are researching- we can argue that "silent" commits can be considered as not contributory. Therefore, we have set a flag for such commits in order to compute them separately or leave them out completely.

[1] A commiter is a person who has write access to the repository and does a commit -an interaction- with it at a given time.

[2] In a versioning system there is actually no file removement. Files that are not required anymore are stored in the Attic and could be called back anytime in future.

[3] Aggregated modules are modules that are shared between other modules. Such modules generally include system-wide administration and scripts. This information is kept in the CVSROOT/modules file.

Write access to the versioning system is not given to anyone. Usually this privilege is only given to contributors who reach a compromise with the project and the project's goals. External contributions -commonly called patches, that may contain bug fixes as well as implementation of new functionality- from people outside the ones who have write access (commiters) are always welcome. It is a widely accepted practice to mark an external contribution when commiting it with an authorship attribution, so we have constructed certain heuristics to find and mark commits due to such contributions.

Once the logs are parsed and transformed into a more structured format, some summarizing and database optimization information is computed and data is stored into a database.

The postprocess is composed of several scripts that interact with the database, analyse statistically its information, compute several inequality and concentration indices and generate graphs for the evolution in time for a couple of interesting parameters (commits, commiters, LOCs...). Results are shown through a publicly accessible web interface that permits an easy inspection of the whole repository (general results), a single module or by commiters. Therefore these results themselves are again available for remote analysis and interpretation by project participants and other stakeholders.

3 General results

General results are those that pertain to the whole CVS repository. The number of modules, commiters and commits can give us an idea of the size of the repository we are analyzing. It may also give some information about the inner structure and organization of the whole project. For instance, in the data presented in Table 1 it can be observed that the mean number of commiters per module is near 1.3, which obeys to the fact that many single-commiter modules exist.

Table 1. General results for the GNOME project

Number of modules	756
Number of commiters	992
Number of commits	1,883,271
Number of files	240,621
Lines added	121,711,566
Lines removed	74,290,346
First commit	1997-11-23
Last commit considered	2003-12-06
Number of days	2,204

There are some aspects that could be also be infered from the data about aggregated, removed, changed and final lines. For instance, the number of aggregated lines is almost three times the number of final lines which means that the source code in the repository is code that has been reviewed conscientiously, being possibly a measure of maturity of the software.

Table 2. Inactivity rate for CVS modules in the GNOME project

Inactive modules in the last year	327	43%
Inactive modules in the last two years	233	31%
Inactive modules in the last four years	86	11%

The availability of all the history of the repository allows to make a fast analysis of the liveliness of modules and developers. Table 2 shows how many modules of the GNOME project are not developed anymore.

Table 3. Inactivity rate for CVS commiters in the GNOME project

Inactive commiters in the last year	488	49%
Inactive commiters in the last two years	348	35%
Inactive commiters in the last four years	108	11%

Analogous to the inactivity rate for modules, an inactivity rate for commiters can be obtained. In Table 3, we can see how there exists a high number of inactive commiters, giving an additional hint at personnel turnover.

4 Results for modules

Table 4. General statistics for the 'Evolution'

Commiters	190
Commits	88,157
Files	5,238
Lines Changed	16,411,471
Lines Added	9,360,719
Lines Removed	7,050,752
First Commit	1998-01-12
Last commit considered	2003-12-05

Several statistics concerning each module of the software system can also be computed. As an example, results from analysing the Evolution module are given. Of special interest seem longitudinal results, for example depicting the evolution in size, participants, or distribution of effort.

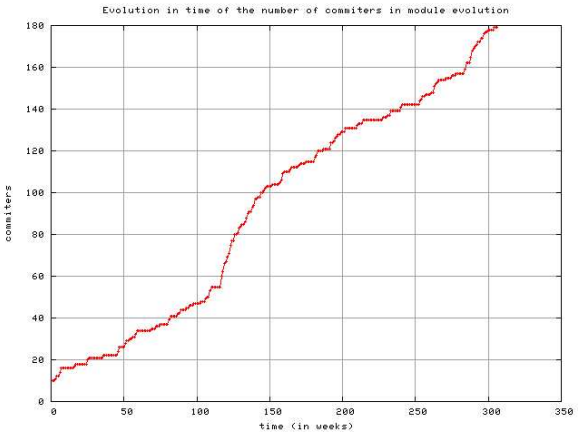

Figure 1. Commiters in time for 'Evolution'.

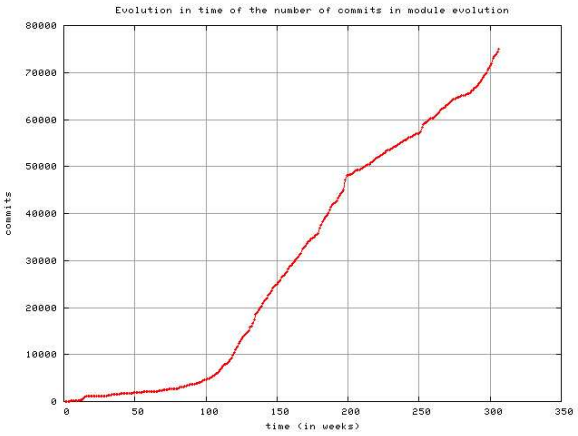

Figure 2. Commits in time for 'Evolution'.

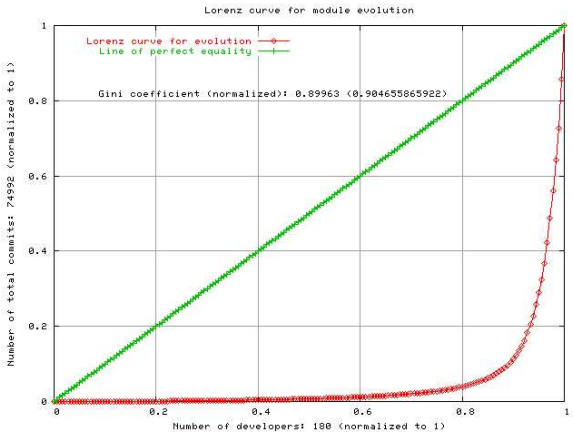

Figure 3. Gini coefficient in 'Evolution'.

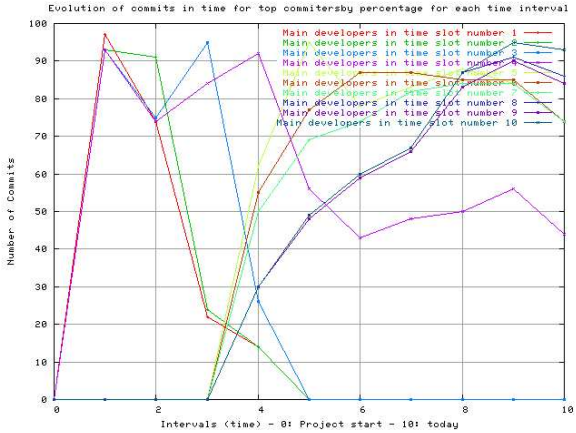

Figure 4. "Generations" in Evolution.

The Gini coefficient given above is used to portray the inequality in the distribution of commits contributed by commiters. In the analogy with economics, commits are usually money and commiters the persons who owns that money; the Gini coefficient would indicate how unequal wealth (in monetary terms) is distributed among the considered persons. The straight curve would show perfect equality, while the Lorenz curve below shows the actual distribution (all normalized to 1). The Gini coefficient gives the area in between [6] and is a simple measure of how unequal the contributions to the project are. Other measures that may give an idea of such characteristics of a project have also been studied, such as the Atkinson index [1] (which is also an inequality dimension as the Gini coefficient) or the Herfindahl-Hirschman index [11], a commonly accepted measure of market concentration that can be used analogously to quantify the concentration of the development work in a development team.

An additional graph shows different generations of commiters during the lifetime of the inspected module. The whole lifetime has been splitted into ten equally long intervals of time. In each time slot, the core group (those developers who lead the development) has been identified as the 20% most contributing commiters during each interval. To every core group in every interval a color is assigned and the evolution in time of that core group is analyzed. Further explanations of this notion can be found in [9]. Generally, this allows to identify several 'generations' of developers that lead a project.

5 Results for commiters

Of course, as the data is available, statistics on single developers can also be computed. These would include the

different and total numbers of modules they are active, the most common filetypes, and of course measures for their participation like commits or changed LOCs. Table 5 shows briefly what kind of data we are able to get from a commiter. Notice that commiters have to have write permission into CVS, so that the analysis of commits in CVS may differ from the one of usual changelogs [3].

Table 5. Statistics for commiter 'acs'

Module	Commits	LOC	First	Last
mrproject	181	5402	02.03.22	02.07.31
libmrproject	39	496	02.03.24	02.07.09

6 Conclusions and further work

As this paper and the presented implementation show, insights into both the current state and the evolution of libre software systems can indeed be gained on a remote basis, even without personal involvement in a project. The information that can be gathered from publicly-available version control systems allows us to have a global perspective of the project and the human resources commited to it not only in present times but also in any point in time since the beginning of the project (or at least the establishment of the source code repository).

This information can also of course be used to try to predict a project's future evolution for control, management and releasing policies [4]. Although some interesting facts on the human resources of these type of projects have been shown as for instance the assumption of 'generations' of leading groups that guide the project temporarily, an enormous research effort should be invested in the near future to gain insight into the dynamics of developer integration into libre software projects. In this sense, there are some proposals that try to use ideas from other knowledge areas as for instance the study and characterization of complex systems [8] and the application of classical (social) network analysis in order to understand them.

In addition, it has to be regarded that software is in any case an important valuable good and that all measurements are key points for the calculation of economic parameters. It has be to noted that if cost estimation is already a problematic task in classical (proprietary) software environments where human and technical resources (and their disposal) are known, in the libre software world this is by far more complex [12]. Any attempt with the aim of solving this lack of knowledge is welcome and having accurate data and information on the process is a good start.

Finally, it should be remembered that the source code repository is not the only public information source available for libre software projects. There exist others that may provide with complementary data. One suite that looks for the integration of software measurement and analysis systems has been proposed by the authors of this paper [16].

References

[1] A. Atkinson. On the measurement of inequality. *Journal of Economic Theory*, (2):244–263, 1970.

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

[3] A. Capiluppi, P. Lago, and M. Morisio. Evidences in the evolution of os projects through changelog analyses. 2003.

[4] J. R. Ehrenkranzt. Release management within open source projects. 2003.

[5] D. Germán and A. Mockus. Automating the measurement of open source projects. Portland, Oregon, 2003.

[6] C. Gini. *On the Measure of Concentration with Espacial Reference to Income and Wealth*. Cowles Commission, 1936.

[7] M. W. Godfrey and Q. Tu. Evolution in open source software: A case study. 2000.

[8] J. M. González-Barahona, L. López-Fernández, and G. Robles. Community structure of modules in the apache project. 2004.

[9] J. M. González-Barahona and G. Robles. Unmounting the "code gods" assumption. Technical report, 2003.

[10] K. Healy and A. Schussman. The ecology of open-source software development. Technical report, University of Arizona, USA, Jan. 2003. `http://opensource.mit.edu/papers/healyschussman.pdf`.

[11] O. Herfi ndahl. *Copper Costs and Prices: 1870 - 1957*. Baltimore: The John Hopkins Press, 1959.

[12] S. Koch and G. Schneider. Results from software engineering research into open source development projects using public data. *Diskussionspapiere zum Tätigkeitsfeld Informationsverarbeitung und Informationswirtschaft*, (22), 2000. `http://wwwai.wu-wien.ac.at/~koch/forschung/sw-eng/wp22.pdf`.

[13] M. Lehman, J. Ramil, P. Wernick, and D. Perry. Metrics and laws of software evolution - the nineties view. 1997.

[14] A. Mockus, R. Fielding, and J. Herbsleb. A case study of open source software development: The Apache server. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*, pages 263–272, Limerick, Ireland, 2000.

[15] E. S. Raymond. The cathedral and the bazar. *First Monday*, 1997. `http://www.firstmonday.dk/issues/issue3\_3/raymond/`.

[16] G. Robles, J. M. Gonzalez-Barahona, and R. A. Ghosh. Gluetheos: Automating the retrieval and analysis of data from publicly available software repositories. 2004.

[17] G. Robles-Martinez, J. M. Gonzalez-Barahona, J. Centeno-Gonzalez, V. Matellan-Olivera, and L. Rodero-Merino. Studying the evolution of libre software projects using publicly available data. Portland, Oregon, 2003.