# Comparison between SLOCs and number of files as size metrics for software evolution analysis

Israel Herraiz, Gregorio Robles, Jesús M. González-Barahona
Grupo de Sistemas y Comunicaciones
Universidad Rey Juan Carlos, SPAIN*
{herraiz, grex, jgb}@gsyc.escet.urjc.es

Andrea Capiluppi
Dipartimento di Automatica e Informatica
Politecnico di Torino, ITALY
Andrea.Capiluppi@polito.it†

Juan F. Ramil
Computing Department
Faculty of Maths and Computing
The Open University, UK
J.F.Ramil@open.ac.uk

## Abstract

*There are some concerns in the research community about the convenience of using low-level metrics (such as SLOC, source lines of code) for characterizing the evolution of software, instead of the more traditional higher lever metrics (such as the number of modules or files). This issue has been raised in particular after some studies that suggest that libre (free, open source) software evolves differently than 'traditional' software, and therefore it does not conform to Lehman's* laws *of software evolution. Since those studies on libre software evolution use SLOCs as the base metric, while Lehman's and other traditional studies use modules or files, it is difficult to compare both cases. To overcome this difficulty, and to explore the differences between SLOC and files/modules counts in libre software projects, we have selected a large sample of programs and have calculated both size metrics over time. Our study shows that in those cases the evolution patterns in both cases (counting SLOCs or files) is the same, and that some patterns not conforming to Lehman's laws are indeed apparent.*

**Keywords:** metrics, software evolution, libre software, empirical studies

## 1. Introduction and aims

Thirty years of research on software evolution have resulted in a set of empirical observations, known as Lehman's Laws of Software Evolution [6]. Although the number of laws has grown from three (in the seventies) to eight (in their latest version [5]), all of them have been empirically proved, by studying projects developed in traditional industrial software development environments.

In recent times, the rise of a new development phenomenon, libre software[1], has opened new horizons to the analysis of software evolution, at least with respect to two issues. The first one is whether the *laws* of software evolution apply to these new environments, where management is loose and contributions from third parties, mainly volunteers, are fostered. The second one derives from the fact that this type of projects make available to researchers a large quantity of public information about the development process which can be retrieved and analyzed. This offers the possibility of having a general view of the landscape instead of just the results of a small number of selected case studies.

Several authors have analyzed the evolution of libre

*The work by the researchers at URJC has been funded in part by the European Commission, under the CALIBRE CA, IST program, contract number 004337, and by the Spanish CICyT under project TIN2004-07296. Israel Herraiz has been funded in part by Consejeria de Educación of Comunidad de Madrid and European Social Fund, under grant number 01/FPI/0582/2005. Part of this work has been developed while Israel Herraiz was visiting The Open University.

†Visiting researcher at The Open University, UK

1. Through this paper we will use the term "libre software" to refer code that conforms to either the definition of "free software" (according to the Free Software Foundation) or of "open source software" (according to the Open Source Initiative).

software, usually considering the Linux kernel as a case study. They have found that the evolution of libre software is different from the evolution of 'traditional' software reported so far. Godfrey and Tu [3, 2] showed that Linux is growing superlinearly in size (using SLOCs as metric). Succi et al. [13] confirmed the super-linearity for the Linux kernel, but found linear growth for GCC and Apache. Recently, Robles *et al.* [9] have repeated the study on the Linux kernel obtaining again super-linear growth (with a greater growth rate), and have also extended the methodology to a wider sample (including the FreeBSD, NetBSD and OpenBSD kernels, and a sample of other 18 large, both in size and in number of contributors, libre software projects), showing in most cases linear or near-to-linear behaviors.

However, there is still a need of more studies, based on wider samples of libre software projects and a larger set of metrics, that could shed some light on the behavior of libre software according to the classical assumptions in software evolution [6]. In addition, most of the studies on libre software evolution have considered SLOC (source lines of code) or LOC as the metric to study, which causes some difficulties in the comparison with the classical works, based on modules or files counts. Therefore, it is also important to state whether both kinds of metrics are comparable with respect to software evolution.

This paper is a first step in this direction. We show that the evolution patterns using SLOC and files counts are quite similar. In addition, we have also found how most of the projects that have been selected show a behavior similar to previous studies on libre software evolution, although some projects in the sample do not. A more particular result is the validation of the conclusions of [3, 2] and [9] about Linux with a different size metric (number of files).

The structure of the rest of this paper is as follows. In the next section the research methodology is explained, including the reasons for choosing the sample of projects to study. In section 3 the main results of applying it to the sample of projects are shown. Section 4 refers some previous work related to this study. Finally, in section 5 we provide some more insight into the results in the form of conclusions.

## 2. Methodology

The methodology we have followed in the present work can be summarized as follows. To begin with, a sample of libre software projects is selected. For each project in the sample, a snapshot of its code (as stored in the SCM repository of the project) at periodic points in time is downloaded and measured. The data obtained is stored in a database which is later queried to build the datasets needed for obtaining the evolution patterns (considering both SLOC and files). These evolution patterns are lated analyzed with detail, to obtain the conclusions of our study. In the rest of this section, some details of the most important aspects of this methodology are provided.

### 2.1. Selection of the sample

A key issue for this study was the selection of the sample of libre software projects to analyze. After considering several chances, we decided that it made sense to focus on some of the largest packages in Debian GNU/Linux.

Debian is one of the largest (and maybe the largest) software systems in the world. Its size is over 229 millions of lines of code, and as some studies have shown [1, 11] it is a healthy system with good reputation of stability and matureness. Debian 3.1, the latest release, contains 8633 packages of source code[2] which are maintained by 1380 developers. These packages are not developed by Debian developers themselves: their work is basically packaging and integrating the upstream software into a coherent GNU/Linux distribution.

Most libre software is developed for Unix systems (in particular, for GNU/Linux), and Debian contains the most demanded and used applications by users, so it is very strange that a well known and used libre software project is not found of Debian (with some exceptions, the most remarkable being some Java-based projects, not included in Debian because of dependencies issues). In other words, libre software which is not part of Debian is the exception, not the rule. Therefore, we have selected the population of software packages of Debian as a representation of the whole population of libre software projects. From this population, we have selected a sample, which satisfies some criteria.

After choosing the base distribution, we selected a sample from it. For that matter, we applied several criteria. The first one is clear: the selected projects must have enough history to make the study. After some discussions, we agreed to select only projects older than 30 months. Otherwise, the amount of data we could retrieve is not enough to make meaningful correlations between the different metrics.

2. *Source* packages, which only include source code. The population of binary packages in Debian contains 15300 packages.

The next criteria is based on the version control system used by the project. For the study, the GlueTheos tool [10] has been used, which is able to download periodical snapshots of source code from a CVS server. Therefore, we limited the sample to projects with a CVS server. Fortunately, CVS is the most popular control version system in libre software projects, and therefore this limitation doesn't imply a significant bias in the sample.

We were also interested in large projects, since they are developed by large groups of developers, thus avoiding the effect of special cases in very small groups of developers. Starting with data obtained for [1], we decided to consider the largest ones.

A result of this selection process, we selected 10 packages, which are, among those largest in Debian, those with a CVS repository and at least 30 months old (the only exception being the Linux kernel, which was included because of its significance in previous studies). In addition, the FreeBSD, NetBSD and OpenBSD kernels were also added to the sample, because these projects are well known large libre software projects and comply with the selection criteria (except for not being part of Debian). The details of the selected projects are shown in table 1. In the first column is shown the name of the project; the CVS modules analyzed is in the second column; the number of SLOCs at the end of the considered period is in the third column; the number of files at the end of the analyzed period in the fourth; the date of the first analyzed snapshot in the fifth; and the date of the last snapshot in the sixth.

In some cases the date of beginning of the study is not the same than the date of the beginning of the project, because we used only the first date with some data in the CVS. With this procedure we had enough data to perform the intended analysis.

### 2.2. Data extraction and analysis

Once the sample of packages was selected, and their CVS repositories was identified, we had to obtain the needed data for each project, and perform the corresponding measurements. For this purpose, we used the already mentioned GlueTheos tool, whose architecture is shown in figure 1.

For each project, the date of the first commit in the CVS was identified, in some cases by using the `history` command available in the CVS client tool, in some others by reviewing directly the CVS web interface of the project. Since the date of the first commit, and with a period of six months, a snapshot of the source code was downloaded.

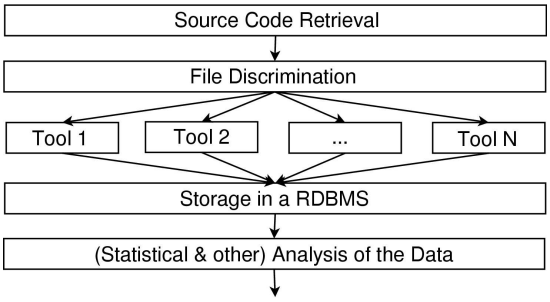


**Figure 1. Architecture of the GlueTheos tool**

After downloading each snapshot, the number of SLOCs and the number of files of source code was measured, using the SlocCount tool [12]. The output of this tool was parsed and stored in a database, that afterwards we queried in order to obtain the number of SLOCs and files over time.

In the case of Linux, since the project does not have a CVS server, the methodology was partially different. The sources of each release were obtained as a `tar.gz` compressed files. They were decompressed and extracted, at the same time that the date of each release was identified. Then, we measured SLOCs and number of files for each release, and obtained the results in the same format than the cases that have a CVS server.

As a result of this procedure, we completed a database with the number of SLOCs and the number of files for each snapshot of each project (taken periodically every six months), except for the case of Linux, where each snapshot corresponds with a release.

The data in the database was analyzed and correlated, leading to the results shown in the next section.

## 3. Results

For each project, the size measured in SLOCs and the size measured in number of files of source code for each snapshot was correlated. The results are shown in table 2. The name of the project is shown in the first column, the linear equation fitting the number of SLOCs and the number of files is shown in the second column, the number of snapshots considered in the regression analysis is in the third column, and the Pearson correlation coefficient in the fourth. It can be seen how all the projects have at least 11 points to make the correlation analysis.

The worst correlation coefficient obtained for those projects is $r^2 = 0.8893$, and the best one is $r^2 = 0.9977$. Therefore we can conclude that, at least in the case of

| Name | Modules | SLOCs | Num. of files | First date | Last Date |
|---|---|---|---|---|---|
| Amaya | `Amaya` | 443295 | 1092 | 1997-06-30 | 2005-05-19 |
| Evolution | `evolution` | 359148 | 1489 | 1997-12-27 | 2005-05-19 |
| FreeBSD | `src` | 1756198 | 5652 | 1993-06-19 | 2005-05-17 |
| Kaffe | `kaffe` | 598387 | 5294 | 1998-04-30 | 2005-05-23 |
| NetBSD | `src` | 2535613 | 13175 | 1992-07-19 | 2005-06-11 |
| OpenBSD | `src` | 1606210 | 6254 | 1997-10-26 | 2005-06-16 |
| Prc-Tools | `prc-tools` | 19176 | 152 | 2000-05-19 | 2005-05-25 |
| Python | `python` | 615113 | 2261 | 1997-06-30 | 2005-05-19 |
| Wine | `wine` | 927415 | 2098 | 1998-12-27 | 2005-05-24 |
| wxWidgets | `wxwidgets` | 1781581 | 4175 | 2002-11-04 | 2005-05-23 |
| XEmacs | `XEmacs` | 2210705 | 5266 | 1997-12-27 | 2005-05-19 |
| XFree86 | `xc` | 2200501 | 6756 | 1997-06-30 | 2005-05-19 |
| Linux | – | 4176875 | 16583 | 1991-09-17 | 2004-12-24 |

**Table 1. Details of the selected projects**

the selected projects, using SLOCs or number of files as size metrics throws the same results, and therefore, for software evolution analysis, the same conclusions can be derived by measuring SLOCs or number of files.

As an example, we show in figure 2 the correlation between the number of SLOCs and the number of files in the case of the Linux kernel, and in figure 3 for the FreeBSD `src` module. It can be seen how the correlation is clearly linear. The rest of the projects follow the same pattern, as the correlation coefficients evidences.

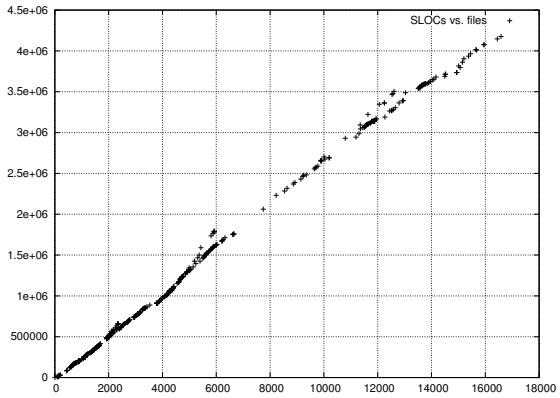


**Figure 2. Scatter plot between the number of SLOCs (vertical axis) and the number of files of source code (horizontal axis) in the case of the Linux kernel.**

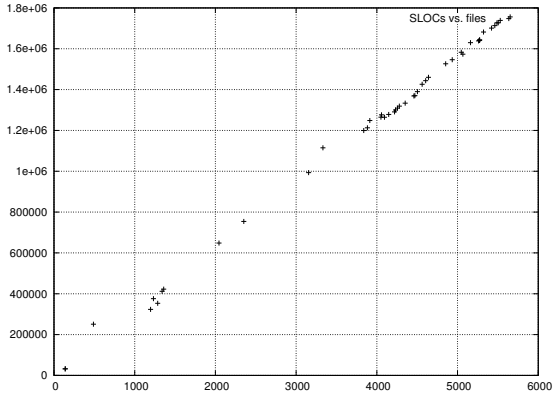


**Figure 3. Scatter plot between the number of SLOCs (vertical axis) and the number of files of source code (horizontal axis) in the case of the FreeBSD `src` module.**

The growth rate for each project was also calculated, with SLOCs and number of files as size metric. For example, in figures 4 and 5, the linear growth over time for FreeBSD `src` module is shown, with SLOCs and num-

| Project | Equation | $n$ | $r^2$ |
|---|---|---|---|
| Amaya | $S = 563.635 \cdot f - 153662.2706$ | 14 | .8893 |
| Evolution | $S = 237.8748 \cdot f - 1397.7322$ | 11 | .9945 |
| FreeBSD | $S = 312.2457 \cdot f + 108.9p28$ | 46 | .9975 |
| Kaffe | $S = 105.7720 \cdot f - 5316.4131$ | 35 | .9842 |
| NetBSD | $S = 189.8916 \cdot f + 101416.3262$ | 43 | .9889 |
| OpenBSD | $S = 219.8261 \cdot f + 230075.6100$ | 12 | .9797 |
| Prc tools | $S = 101.2863 \cdot f + 2347.8471$ | 122 | .9709 |
| Python | $S = 335.8650 \cdot f - 189998.4555$ | 13 | .9552 |
| Wine | $S = 514.9345 \cdot f - 142849.9740$ | 11 | .9927 |
| wxWidgets | $S = 465.9170 \cdot f - 233395.2761$ | 29 | .8950 |
| XEmacs | $S = 424.4406 \cdot f - 26183.4515$ | 15 | .9956 |
| XFree86 | $S = 326.6894 \cdot f - 70754.4803$ | 12 | .9861 |
| Linux | $S = 265.3134 \cdot f - 26979.9934$ | 533 | .9977 |

**Table 2. Correlation between SLOCs and number of files in the selected projects ($S$ is number of SLOCs and $f$ is number of files of source code).**

ber of files as size metrics, respectively. We correlated the number of SLOCs and number of files versus time, with a quadratic equation. We took the coefficient of the quadratic term as a measurement of the growth rate. We are interested only in the sign (positive, negative or zero) of the rate, and not in its exact value (although we have calculated the value of the growth rate, using both SLOCs and number of files as size metrics).

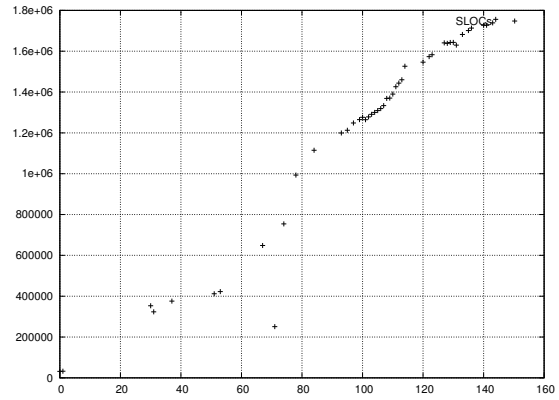


**Figure 4. Growth of FreeBSD `src` module (SLOCs, vertical axis) over time.**

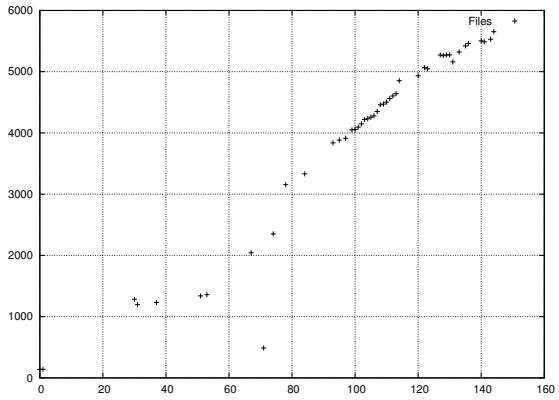


**Figure 5. Growth of FreeBSD `src` module (number of files, vertical axis) over time.**

With these data, the projects were classified in three categories: those whose rate is positive (superlinear growth), those whose rate is nearly zero (linear growth) and those whose rate is negative (sublinear growth). All the projects came under the same category, independently of the selected metric. The results are shown in

table 3. For classifying each project in each category, we took also a look to the evolution graphics, in order to check if the pattern (linear, sublinear, superlinear) was correct (for example, see figures 4 and 5).

Focusing in the case of Linux, we can conclude that the studies by Godfrey [3] and Robles [9] are also verified using number of files of source code instead of SLOCs, as it is shown in figure 6. In the vertical axis, number of files of source code are shown; in the horizontal axis, time. Each curve represents a version of the Linux kernel.

The quadratic correlation throws a Pearson coefficient of $r^2 = 0.9325$ when using release number as time unit. If we use instead days since the first release, the coefficient is $r^2 = 0.9066$. In this last case, the equation is $S = 0.7918 \cdot 10^{-4} \cdot t^2 - 0.4002 \cdot t + 143.1589$, where $S$ is the number of files of source code, and $t$ is the time in days.

As it is shown in table 3, only three of the projects in the sample grow with a sublinear pattern. In other words, only these three projects grow with a pattern similar to that described in [6]. Therefore, we can also conclude that most of the analyzed projects do not conform to the predictions of the Lehman's fourth law.

## 4. Related work

To our knowledge the first reference showing new growth patterns in the case of libre software is [3]. Godfrey showed that the growth of Linux followed a quadratic law, which is clearly different from the classical assumptions about the evolution of software systems, formulated by Lehman [6], which postulated sublinear growth. Lehman himself (*et al.*) noticed this case, and labeled up it as an anomaly [8], proposing as possible causes that an open pool of developers could be contributing code to Linux, and furthermore that a high level of code cloning could be present in the code.

Robles *et al.* [9] repeated and updated the study of Godfrey about the Linux kernel, and found that the growth rate has even increased in the last five years. Therefore, the growth would be not only superlinear, but even more superlinear as time passes.

Other software evolution studies [4] have found the same superlinear pattern, considering libre software projects in SourceForge. Despite measuring number of files as well as SLOCS, the conclusions about the growth patterns in those studies were based only in SLOCs or LOCs measurements. However, in previous studies, Lehman *et al.* had defended that SLOCs is not the best metric for software evolution analysis[7], using instead modules or files in his research, in which was followed by most authors.

With respect to the distribution used as a base for the sample of projects selected for this study, it can be said that Debian is a well known GNU/Linux distribution. It has been used as a case study since some years ago. As an example, the most recent reference studying Debian is [1]. These studies have focused in analyzing Debian from a high level point of view, making aggregated analysis of the whole distribution, rather than focusing on each package, and analyzing it in a deeper way.

## 5. Conclusions

Libre software is being studied with increasing interest since some years ago. Some of the early research in this field already showed that the evolution patterns in libre software programs are different to those found in the classical studies. Specifically, non-sublinear growth patterns were reported. Later research seemed to confirm those findings, but still with some difficulties in the comparison of results. In particular, all these studies about libre software were based on SLOCs or LOCs as size metric, while traditional studies considered modules or files.

In this work, we have revalidated the finding about the non-sublinear growth pattern with some more examples. Moreover, we have shown that these results are the same with two different size metrics (SLOCs and number of files). At least in the case of the selected sample of projects, we can conclude that the number of SLOCs is a metric as good as the number of files for software evolution analysis.

However, we can still not explain at this moment why libre software projects are growing with non-decreasing rates. Deeper analysis are needed, measuring more features of the software (such as complexity), and looking for hot points of evolution by means of architectural analysis over time. It is very interesting to try to solve how these projects are controlling the complexity (if they are). The control of the complexity seems to allow them to grow in size in a fast way, and with an increasing rate. It would be also interesting to try to find dead parts which are no longer evolving, and if there are cloned parts in the architecture.

Moreover, effort measurements are crucial to find out if it is true that an open pool of developers is contributing to the evolution of libre software, and how large these contributions are.

In any case, it seems clear that there must be limits for these parameters. Size can not grow forever, as

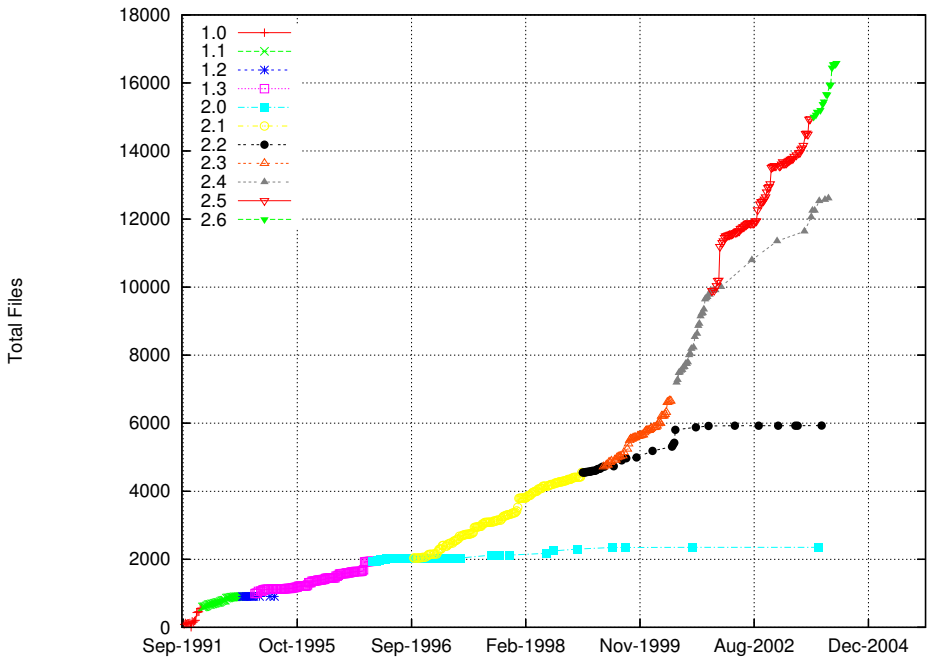


**Figure 6. Evolution of the size of Linux kernel (vertical axis, as number of files of source code) over time.**

| Project | Growth rate (SLOCs) | Growth rate (# files) | Category |
|---|---|---|---|
| Amaya | 1.45 | -0.0055 | Linear |
| Evolution | -31.89 | -0.17 | Sublinear |
| FreeBSD* | 15.16 | 0.056 | Linear |
| Kaffe | 77.13 | 0.71 | Superlinear |
| NetBSD* | 152.74 | 1.04 | Superlinear |
| OpenBSD* | 401.20 | 2.01 | Superlinear |
| Prc tools | 4.31 | 0.044 | Superlinear |
| Python | 18.43 | -0.062 | Linear |
| Wine | 50.06 | 0.064 | Linear |
| wxWidgets* | 587.56 | 0.29 | Superlinear |
| XEmacs | -259.44 | -0.60 | Sublinear |
| XFree86 | -412.28 | -1.47 | Sublinear |
| Linux* | 186.21 | 0.71 | Superlinear |

**Table 3. Growth pattern for each project. Growth rates in size per each six months (projects with * show rates per month rather than per six months)**

complexity makes more difficult to add functionality to a project, and causes it to slow down. Developers can not improve their productivity forever, as an increasing number of developers working in the same artifact provokes inefficiencies.

Therefore, more studies are needed to enlighten some of this doubts, and to better understand the fast growth of libre software. With time, even the formulation of a theoretical framework to explain this behavior should be possible.

## References


[1] J. J. Amor-Iglesias, J. M. González-Barahona, G. Robles, and I. Herraiz. Measuring libre software using debian 3.1 (sarge) as a case study: Preliminary results. *Upgrade*, VI(3), June 2005.

[2] 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.

[3] M. W. 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.

[4] S. Koch. Evolution of open source system software systems - a large scale investigation. In *Proceedings of the First International Conference on Open Source Systems*, 2005.

[5] M. Lehman, J. Ramil, P. Wernick, and D. Perry. Metrics and laws of software evolution - the nineties view. In *Proceedings of the Fourth International Software Metrics Symposium*, 1997.

[6] M. M. Lehman and L. A. Belady, editors. *Program evolution: processes of software change*. Academic Press Professional Inc., 1985.

[7] M. M. Lehman, D. E. Perry, and J. F. Ramil. Implications of evolution metrics on software maintenance. In *ICSM*, pages 208–218, 1998.

[8] M. M. Lehman, J. F. Ramil, and U. Sandler. An approach to modelling long-term growth trends in software systems. In *Software Maintenance, 2001. Proceedings. IEEE International Conference on*, pages 219–228, 2001.

[9] G. Robles, J. J. Amor, J. M. Gonzalez-Barahona, and I. Herraiz. Evolution and growth in large libre software projects. In *Proceedings of the 8th International Workshop on Principles of Software Evolution*, pages 165–174, Lisbon, September 2005. IEEE Computer Society.

[10] G. Robles, J. M. Gonzalez-Barahona, and R. A. Ghosh. Gluetheos: Automating the retrieval and analysis of data from publicly available software repositories. In *Proceedings of the International Workshop on Mining Software Repositories*, pages 28–31, Edinburg, Scotland, UK, 2004.

[11] G. Robles, J. M. Gonzalez-Barahona, and M. Michlmayr. Evolution of volunteer participation in libre software projects: evidence from Debian. In *Proceedings of the 1st International Conference on Open Source Systems*, Genova, Italy, July 2005. To appear.

[12] Sloccount. `http://www.dwheeler.com/sloccount/`.

[13] G. Succi, J. Paulson, and A. Eberlein. Preliminary results from an empirical study on the growth of open source and commercial software products. In *EDSER-3 Workshop, co-located with ICSE 2001*, Toronto, Canada, May 2001.