Towards a theoretical model for software growth*

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

Abstract

Software growth (and more broadly, software evolution) is usually considered in terms of size or complexity of source code. However in different studies, usually different metrics are used, which make it difficult to compare approaches and results. In addition, not all metrics are equally easy to calculate for a given source code, which leads to the question of which one is the easiest to calculate without losing too much information. To address both issues, in this paper present a comprehensive study, based on the analysis of about 700,000 C source code files, calculating several size and complexity metrics for all of them. For this sample, we have found double Pareto statistical distributions for all metrics considered, and a high correlation between any two of them. This would imply that any model addressing software growth should produce this Pareto distributions, and that analysis based on any of the considered metrics should show a similar pattern, provided the sample of files considered is large enough.

1 Introduction

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

In fact, many different metrics for size and complexity do exist, and have been successfully used in many empirical studies. However, due to this diversity in metrics, comparison of results is not always easy, and the basic question of which metrics are enough to understand a certain aspect of the source code of a project is still largely unsolved. For obtaining some insight in both issues, we have studied a large quantity of source code (about 700,000 C files) corresponding to mature, stable software in use in a Unix-like software distribution, FreeBSD.

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

FreeBSD is of course not the only large collection of libre software: there are several other projects gathering software from many different libre software projects, adapting it, and producing an integrated system, ready to be used. Among them, several Linux distributions (Debian, Fedora, Ubuntu, Mandriva, etc.) are the most well known. In our case, we have selected FreeBSD because it is a good compromise between quantity of code and simplicity in packaging. In FreeBSD, source code packages (called *ports*) are easy to retrieve and handle automatically.

With the quantity of source code in FreeBSD (more than 1.7 millions of files, and 400 MSLOC in total, more than 40% of them corresponding to C code), it is possible to apply statistical methods to find out correlations and patterns in the set of analyzed data. In the case of this paper, our first motivation was to find out which independent metrics may be used to characterize size and complexity. To our surprise, we have also found that all metrics considered follow a statistical distribution (double Pareto) that has been deeply studied in other fields.

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

*This work has been funded in part by the European Commission, under the FLOSSMETRICS (FP6-IST-5-033547) and QUALOSS (FP6-IST-5-033547) projects. 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.

The rest of the paper is as follows. Next section reviews some previous work related to the study. Third and fourth sections describe the data sources and the methodology used. Then, main results and findings are presented, including a discussion on the suitability for modeling software growth of some models based on the statistical distributions found in our study, which are used in other domains. Finally, some conclusions and directions for further research are offered.

2 Related work

In 2000, Godfrey [5] pointed out that the Linux kernel was growing with a pattern which did not corresponded to Lehman's laws [10]. In a latter work, Lehman [9] qualified the case of Linux as an anomaly. Furthermore, some claimed that the studies were not comparable, because they used different metrics (Godfrey used SLOC, Lehman number of modules, which are actually number of source code files not counting header files).

This case shows how the issue of which metrics are suitable to empirically study a software project is important. Traditionally, SLOC has been used in the empirical studies of libre software projects [13, 8], while LOC, number of files, or some complexity modules are common in other domains.

Some studies have specifically addressed the problem of comparable metrics. For instance, a previous work of the authors of this study [6] found that in the context of software evolution the number of source code files and SLOCs were highly correlated. However, that analysis was performed measuring the total quantity of software in the SCM repository of each project every six months, and then correlating all the points obtained, which means that the studied files were not internally independent (we studied different versions of the same files). To avoid this bias, the study presented here has performed measures on a collection of software at a single point in time. The code included in that collection is not composed of different versions of the same files, but correspond to unrelated files, in many cases written by different developers, and can be considered statistically independent. Therefore, if some dependencies are found among the metrics in this case, we can conclude that it is not because of any internal relationship among the files.

After collecting the metrics for the whole collection, we found that all of them showed determined statistical distribution. This finding is not new, and has been noted in other fields. For instance, Mitzenmacher [11] describes some phenomenons that develop different statistical distributions. Among them, the distribution of the size of files in a tree of a filesystem is a *double Pareto distribution*. This is the same distribution that we found for the metrics of our set of projects. It has been also found in similar domains, such as in the files available in web and FTP servers [2]. Mitzenmacher has proposed a theoretical model to explain why the distribution of the size of files follows double Pareto [12], which uses concepts quite close to those found in the regular working of a source control system. Therefore, we suggest that it could be easily adapted to simulate and forecast the growth of software.

The problem of obtaining a model for software growth has also been addressed before. For instance, Turski [15, 16] developed a model based on Lehman's laws. Unfortunately, it is not verified by the growth pattern of some libre software projects. Some other models[14, 4, 1] have tried to characterize the process of software development in the libre software world, but they have not been yet checked against the actual history of a meaningful quantity of software projects.

3 Data sources

All files in the study were obtained from the source code available in the *ports* of the FreeBSD operating system. 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.

FreeBSD[1] is developed as a complete operating system. The kernel, device drivers and all *userland* utilities (such as the shell), are maintained in the same source code management system, and external applications and utilities are packaged by the same set of developers. As a contrast, in the case of GNU/Linux the kernel is developed by one group of developers, *userland* utilities and applications by several others (such as the GNU project) and everything is packaged together still by others (to compose so-called Linux distributions).

In FreeBSD, a *port* is a form of package, including source code for an external application or library. It contains, among other files, a makefile, which is able to download the source code from the original web or FTP site, to apply patches developed by the *porters* of FreeBSD (if any), to compile, and to install it into a live system[2]. In a typical FreeBSD system, ports are stored in */usr/ports/*. Among the available targets implemented by the makefile, we find *fetch*, which obtains the original source code and the patches, and stores the result in a directory (typically, */usr/ports/distfiles*).

We invoked this target for a collection of $16,037$ ports in a FreeBSD 6.0-RELEASE system, on an AMD64 platform. The retrieval of the source code happened during December

[1]More information about FreeBSD can be found at http://en.wikipedia.org/wiki/FreeBSD

[2]More information can be found at http://en.wikipedia.org/wiki/FreeBSD_Ports

2006. From these ports, we measured just $13,116$, because the rest did not contain source code (for instance, some ports contain documentation). After uncompressing, we obtained a set of $1,700,927$ files.

4 Methodology

In order to identify the programming language of each file, and its size in SLOC, we used the SLOCCount tool[3]. For this study, we selected only files written in C. We used the *metrics* set of tools[4] to measure the rest of the considered metrics, in addition to the standard *wc* Unix tool (to measure LOC), and *exuberant-ctags* to measure the number of functions. The result of the measurement process was a list of files, each one annotated with the set of metrics measured for it.

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

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

We repeated the same procedure, but this time for the logarithm of the metrics. The correlation coefficients showed strong relationships among the logarithm of the metrics. Moreover, the statistical distributions appeared to be very close to a normal distribution.

4.1 Selected metrics

We have measured size and complexity for all the files contained in FreeBSD that were written in C. For these two variables we considered different metrics, shown in table 1.

To measure size we considered:

- Source Lines of Code (SLOC)
  We used the definition given in [3], which is:

  > 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

  We decided to obtain this metric because it has been traditionally used in the study of the evolution of libre (free / open source) software projects (as an example, we cite [5, 13]).

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

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

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

- Number of comments
  We measured the number of comment blocks. If a comment occupied several lines, but it was only one block, it was considered only one comment.

- Number of C functions
  We counted the number of functions inside the file. To count functions, we used the tool *exuberant-ctags*, combined with *wc*. We decided to add also a semantical metric, in the hope of finding some difference between syntactical and semantical metrics.

To measure complexity we considered:

- McCabe's cyclomatic complexity
  We used the definition given in [7]. McCabe's cyclomatic complexity is the classical graph theory cyclomatic number, indicating the number of regions in a graph.

- Number of function returns
  The tool used to measure McCabe's cyclomatic complexity was able to measure also this metric. Compared again to graph theory, the number of function returns gives us the number of exit points of the graph formed by the number of paths that the execution can follow inside the function.

- Halstead's length, volume, level and mental discriminations
  From the Halstead's Software Science we selected these two metrics. The exact definition is also available in [7].

A summary of the metrics selected, and the symbols used in the rest of the tables of this paper, is shown in table 1.

[3]Available at http://www.dwheeler.com/sloccount
[4]Available at http://libresoft.urjc.es/Tools/metrics

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

Table 1. Selected metrics for the study

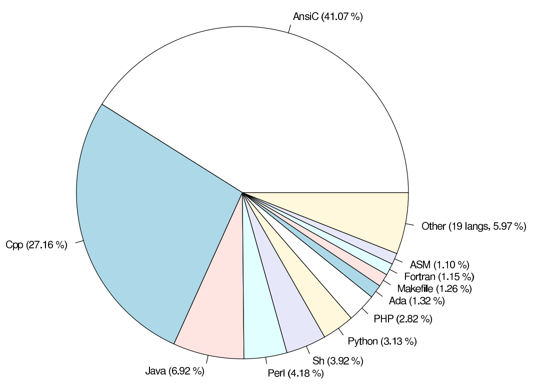

Figure 1. Distribution of programming languages. Values are percentage of source code files in the sample.

5 Results

We measured $1,700,927$ files, distributed by programming language as indicated by figure 1. The indicated percentages are of files written in that language compared to the total number of source code files. C, C++ and Java account for more than the three quarters of the whole sample.

These $1,700,927$ source code files contained $409,542,955$ SLOC. The files were written in 30 different programming languages. Five of them (C, C++, Shell, Java, Perl and Python) covered 88% of the files and 83% of the SLOCs.

The average size of the files was 241 SLOC, with a standard deviation of $1,199$. The largest file contained $761,870$ SLOC ($761,883$ LOC), and the smallest file 0 SLOC (ranging from files with 0 LOC to $5,729$ LOC).

From the whole sample of files, $698,506$ were written in C language, from which $455,366$ files were non-header files.

We could not measure $3,304$ files, because our tools did not correctly parse the source code. We found also 162 empty files, that were removed from the sample. Therefore the final sample for all the files written in C language contained $695,039$ files.

In the sample we found some files with very high values for some metrics. After inspecting those files, we found many that were not written by humans but automatically generated. Those files are outliers (observations that deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism). In our case, this different mechanism is automatic generation (compared to the usual mechanism of generation by a human developer). As we are interested in the process of software development, we did not consider those files that are not properly part of this process.

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

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

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

After testing all the metrics, all the distributions resulted to be very close to a normal distribution. As an example, we show the test for the distribution of the logarithm of SLOC in figure 2. All the points follow a straight line, although values in the tails seem to deviate from a straight line. In any case, we can consider this distribution to be normal (as the low value of the kurtosis, table 2, evidences as well). We only show 1000 randomly extracted points, because the graph with all the 695039 points occupied to much memory as to be included in the electronic version of this paper.

However, this normality test is not robust. It could happen that the distribution is not lognormal but a power law

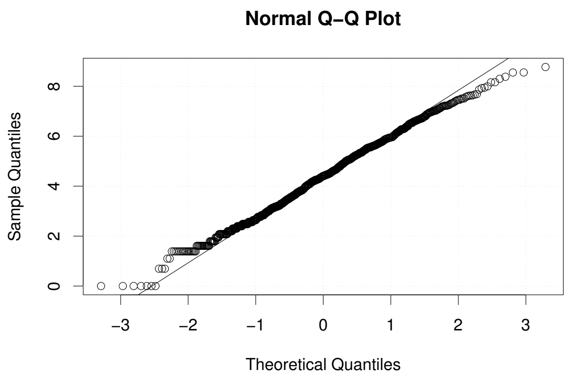

Figure 2. Quantile-Quantile plot for the distribution of the logarithm of SLOC.

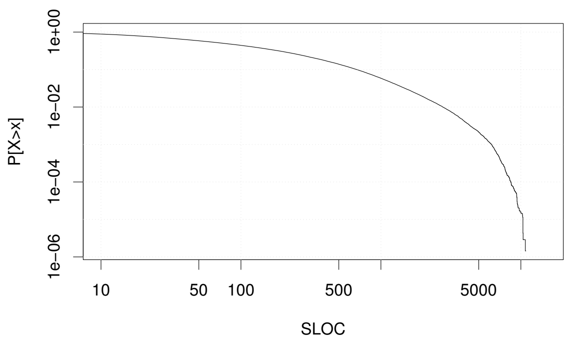

Figure 3. Complementary cumulative distribution function of SLOC. Logarithmic scale.

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

Figure 3 shows the same conclusions that figure 2. The body of the distribution presents a lognormal behavior (the curved transition point between the two tails is typical of the lognormal distribution), but the tails deviate from the normality, and are closer to a power law behavior (the straight lines in the tails are typical of the power law distribution). Therefore, in our case we have a mixed behavior. This mixed distributions has been found before [12]. It is called a *double Pareto distribution*. The tails for low and high values are straight, and there is a curved transition point (or set of points) where the two straight tails connect. The power law would appear in figure 3 as straight line, and the lognormal distribution would not have straight tails on the extreme values.

As the distribution of the logarithm of the metrics were normal (or more properly, very close to normal), we were able of applying the three sigma rule. In the case of LOC, the standard deviation of our sample was 1.45, and the mean was 4.95. Therefore we can consider those files falling below $4.95 - 3 \cdot 1.45 = 0.60$ and over $4.95 + 3 \cdot 1.45 = 9.30$ in the lognormal distribution to be outliers. These values, transformed back to LOC are $e^{0.6} = 1.82$ LOC and $e^{9.30} = 10,938$ LOC. We did not remove those files containing only 1 LOC, because they were not too common in our sample and probably not automatically generated. Certainly, we also found files over $10,938$ LOC non automatically generated (this is, at a deep look apparently written by a human developer). But most of files over that value were automatically generated, and the non automatic files were the exception. Therefore we decided to remove those files. This supposed to remove 979 files. The final sample contained then $694,060$ files.

We repeated the same plots that in figures 2 and 3, and we did not find any difference between the original and the final sample. So the removal of outliers did not affect the shape of the statistical distribution of the sample. In any case, for instance figure 3 shows clearly that the influence of values was very low (the probability of having files over $10,000$ LOC was around 10^{-6}). So we only removed a few points in the right tail, with not very much influence in the distribution.

The properties of the final samples for all the metrics are shown in table 2. As the difference between the mean and the median shows (and also the value of the skewness), the distributions were right tailed. When transformed to logarithm, all the distributions became normal. In that table, the values for the kurtosis are shown both for the original sample and for the logarithm of the sample. An ideal normal distribution has a kurtosis of 0. The closer the value is to 0, the closer the distribution of the sample is to a normal distribution. Therefore, all the distributions of the logarithm of the metrics are close to a normal distribution.

We also examined some composed metrics: number of blank files per LOC and SLOC, number of comments per LOC and SLOC, and number of comment lines per LOC and SLOC. In the case of these composed distributions, we obtained also double Pareto distributions. For instance, figure 4 shows the Quantile-Quantile plot for the distribution of logarithm of the number of comment lines per SLOC. The distribution is composed for a main body that is normal (along the straight line in the plot), with low and high values tails deviating from normality. The same behavior may be observed on the plot of the complementary cumulative distribution function, in a logarithmic scale. Figure 5 shows that plot; there two straight tails connected by a curved segment. Again, this is the typical profile of a double Pareto distribution.

Metric	Mean	Median	Std. dev.	Variance	Kurtosis	Kurtosis (log)	Skewness	Min.	Max.
SLOC	264	76	556	309006	47.55	-0.47	5.67	0	10860
LOC	380	131	738	544920	42.43	-0.02	5.40	1	10900
FUNC	7	1	17	287	547.29	-0.75	11.85	0	2116
BLKL	48	16	98	9539	95.99	-0.38	6.70	0	6885
CMTL	65	29	137	18818	167.77	$+0.69$	8.75	0	9928
CMTN	31	8	97	9462	1068.46	-0.47	23.13	0	9473
CYCLO	41	6	103	10568	85.80	-1.05	6.97	1	4283
RETUR	10	1	28	809	236.49	-0.57	10.50	0	2105
HLENG	1519	392	3575	12785718	81.00	-0.44	7.21	2	77190
HVOLU	12770	2507	34903	1218214882	118.23	-0.27	8.59	2	1075000
HLEVE	0.0784	0.0318	0.1736	0.0302	71.64	-0.24	7.51	0.0000	2.0000
HMD	2770012	77711	25587476	$6.55 \cdot 10^{14}$	4081.83	-0.33	55.83	1	2147483647

Table 2. Descriptive statistics of the sample of files written in C (694060 files). Kurtosis shown both for the original sample and for the logarithm of the sample.

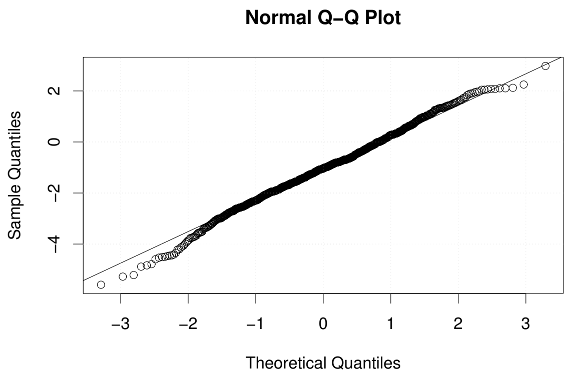

Figure 4. Quantile-Quantile plot for the distribution of the logarithm of the number of comment lines per SLOC.

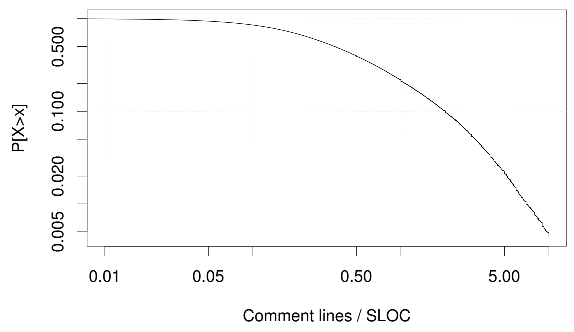

Figure 5. Complementary cumulative distribution function of the number of comment lines per SLOC. Logarithmic scale.

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

Table 3 include all the Pearson coefficients of the correlations among the logarithm of all the metrics. Only the decimal part is shown if the coefficient is not equal to 1. The matrix is symmetrical, so only values under the diagonal are shown. If we take for instance SLOC, we can see that all the correlation coefficients are very high. There are some exceptions, when correlating with blank lines, comments or comment lines. This is a logical conclusion, because those metrics are supposed to be independent (although there is some degree of correlation, for example a large file is likely to contain more comments than a small file). As an example, figure 6 shows the scatter plot of Halstead's length against SLOC, in logarithmic scale. The correlation between the two metrics is clearly shown. Only $10,000$ randomly extracted points are shown, because the graph with the nearly $700,000$ files occupied to much memory as to be included in the electronic version of this paper.

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

	SLOC	LOC	FUNC	BLKL	CMTL	CMTN	CYCLO	RETUR	HLENG	HVOLU	HLEVE	HMD
SLOC	1											
LOC	9844	1										
FUNC	8216	8187	1									
BLKL	9186	9364	8203	1								
CMTL	6742	7676	5477	6903	1							
CMTN	7612	7991	6018	7479	7877	1						
CYCLO	9325	9164	7856	8510	6300	7105	1					
RETUR	7496	7469	7498	7057	5360	5813	7674	1				
HLENG	9817	9655	7942	8999	6530	7450	9146	7219	1			
HVOLU	9815	9652	7921	9002	6532	7431	9123	7192	9994	1		
HLEVE	9033	8851	7005	8207	6157	7016	8823	6714	8256	9198	1	
HMD	9715	8882	7737	8882	6519	7420	9187	7154	9914	9897	9666	1

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

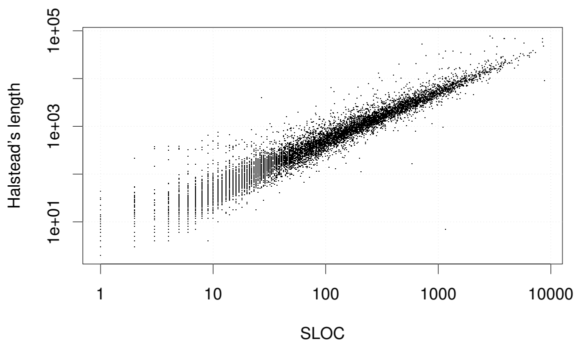

Figure 6. Halstead's length vs. SLOC scatterplot. Logarithmic scale.

6 Towards a theoretical model for software evolution

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

Double Pareto distributions have been found in other fields. For instance, the distribution of the size of the files in a filesystem or tree, follows a double Pareto distribution. For this case, Mitzenmacher [12] proposed the *Recursive Forest File Model*, which is the generalization of others that tried to explain why the size of files follows a double Pareto distribution.

In that model, each tree in the *forest* begins with an initial node of size 1. Then, in each step, a node is randomly selected. From that node, a new node is created as a child. Each one of the edges is labeled with a value. The size of a file is computed as the sum of the edges going from the initial node to the considered node. This would be easily adapted to the case of a version control system. The initial node would be the initial revision of a file. Each step, would be a new revision. If in a node more than one child appears, then all the nodes but one corresponds to branches parallel to the main trunk. The edges would be changes between revisions, and would be labeled with the size of that change. If a file is deleted, that node disappears from the tree. It would correspond to a revision where the file is deleted.

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

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

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

7 Conclusions and further work

We have analyzed the source code of all the packages in the FreeBSD operating system. We retrieved the source code using the source packages of the system (*ports*). This meant to obtain 1.7 millions of files, with a total size of 410 MSLOC. From this set of files, 700, 000 files were written in C. We measured size and complexity, using different metrics, for all these files.

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

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

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

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

The most interesting model we have found is the *Recursive Forest File* model, proposed by Michael Mitzenmacher [12]. It can explain how files change over time, and how they are inserted and removed from a tree of files. Because of its nature, it would be easily adaptable to the case of a source control system, being therefore able of explaining how and why software grows.

In further research, we will try to adapt this model to the case of a control version system, and to verify it against the actual history of a software project.

References

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

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

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

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

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

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

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

[8] 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.

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

[10] 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.

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

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

[13] 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.

[14] 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.

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

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