Free/Libre and Open Source Software Metrics

Sponsored through Framework Programme Sixth (Call 5) by

Document Information

Version: 1.0
Date : Nov 15, 2009
revision: 1

Owning Partner: WUW

Author(s):
Stefan Koch
Santiago Dueñas
Israel Herraiz
Daniel Izquierdo
Ioannis Stamelos
Ioannis Samoladas
Sulayman K. Sowe
Kirsten Haaland
Rishab Ghosh

Reviewer(s):
Jesus M. Gonzalez-Barahona

To:
Public

Purpose of distribution:

The FLOSSMetrics Consortium consists of: Universidad Rey Juan Carlos, University of Maastrich, Wirtschaftsuniversitaet Wien, Aristotle University of Thessaloniki, Conecta s.r.l., Zea Partners and Philips Medical Systems PMS Nederland B.V.

Printed on at

Status:		**Confidentiality:**	
[ ]	Draft	[x] Public	- Intended for public use
[ ]	To be reviewed	[ ] Restricted	- Intended for FLOSSMETRICS consortium only
[ ]	Proposal	[ ] Confidential	- Intended for individual partner only
[x]	Final/Released		

Deliverable ID: D5.1

Title:

High Level Studies

Deliverable: D5.1

Title: High Level Studies

Executive Summary:

This report details the focussed studies which were undertaken using the FLOSSMetrics database and data set. They do not intend to conform a comprehensive analysis of FLOSS development, but a set of examples of how the data collected by FLOSSMetrics can be used for research studies. All the studies performed have in common some context related to software development. However, they make use of different sources of information, and in some cases they aggregating results from several of those different sources. For each study, a standardised format is used for a quick description, in order to ease accessibility and comparisons. Whenever possible, in-depth information is also provided for the studies, especially on context and motivation, as well as methodology and results.

CHANGE LOG

Ver.	Date	Author	Description
0.1	24/07/2007	Stefan Koch	Initial proposal for structure
0.2	14/08/2008	Gregorio Robles	Minor modifications
0.3	02/02/09	Stefan Koch	Integration, Consolidation
0.4	16/02/09	Jesus M. Gonzalez-Barahona	Minor editions, some review
0.5	25/03/09	Stefan Koch	Minor changes, redesign for preliminary submission
0.6	29/04/09	Stefan Koch	Preliminary results
0.7	01/04/09	Santiago Dueñas	Final version for the review
0.8	01/04/09	Jesus M. Gonzalez-Barahona	Final review
0,9	22/07/09	Stefan Koch	Preparations for final version
0.91	10/11/09	Santiago Dueñas	Added HLS from partners
1.0	15/11/09	Jesus M. Gonzalez-Barahona	Review and Release

APPLICABLE DOCUMENT LIST

Ref.	Title, author, source, date, status	Deliverable Identification

TABLE OF CONTENTS

1. Introduction

1.1 Objectives

The main objective of this report is to show evidence that demonstrates the possibilities for focussed studies based on the data provided by FLOSSMetrics. With the data obtained in WP3 (Database workpackage), the classification scheme from WP4 (Analyses workpackage), and the developed database, the FLOSSMetrics project developed several in-depth studies. These studies have the ultimate goal of showing the usefulness of the obtained data for advancing the knowledge base on libre software engineering in several fields, especially software engineering, but also focussing on merging results and insights with economics or management.

The studies presented in this report should be considered just as examples of cases that can be performed with the FLOSSMetrics database, since many others are possible. Therefore, the main aim of this deliverable is to highlight the convenience of providing this kind of aggregated data source to the scientific community. Of course, any insights gained from the studies themselves will also help in advancing the state of the art in research in libre software engineering, and provide additional stimulus for the scientific community. Using these studies as starting point, or even as templates for specific studies, researchers or research groups will find it easier to start working with the information which is provided to them by FLOSSMetrics.

1.2 Structure of the Deliverable

For each study, a standardised format is used for a quick description, in order to ease accessibility and comparisons. The first part of this report lists all studies in this common format. This format includes among other information, the different data sources used, motivation and methodologies.

In the second part of this report, whenever possible, in-depth information is provided, especially on context and motivation, as well as methodology, results and conclusions.

The annexes, delivered as a separate document, include some papers (usually submitted to some scholarly workshop or conference) with presentations related to some of the studies. Those papers are included just to provide some context on the kind of studies being performed. Not all the effort devoted to those papers (and the underlying studies) has been a part of FLOSSMetrics, in several of them, collaboration with researchers not in the project has been the base for the study.

2. Focused Studies – Overview

2.1 On the Validity of The Laws of Software Evolution

- **Expected Outputs/Results**
  For characterising the past development, growth rates can be computed for several aspects. This should help to determine the speed in which the activity within the lifetime of the project has taken place.
  Growth rates can be computed for:
  - Source code (constituting a study of software evolution)
  - Programmers
  - Bugs
  - Bugs reporters
  - Mailings

- **Data sources used**

Souce Code Management Systems	Yes
Mailing Lists	Maybe
Issue Tracking Systems	Maybe
Other (specify)	None

- **Methodology**
  Within FLOSSMetrics, prior studies will be replicated and refined. For computing and characterising the growth rates, the following methodology will be adopted. This is taken from a prior study of one of the project participants. Note that the description pertains to growth in size (source code evolution), but is to be applied in similar terms for the other aspects of interest.
  The first step is to analyse whether a linear or other growth pattern is present in the data. To this end, both a linear and a quadratic model are computed for each project, taking the size in lines-of-code *S* as a function of the time in days since the first commit *t*, which is used as project start date, and using one month as time window. Therefore models A and B were formulated simply as:

$$S_A(t) = a * t + b$$
$$S_B(t) = a * t^2 + t * b + c.$$

- **Filtering and other techniques applied on the data**
  - Size in lines-of-code as a function of the time in days

- **References**
  - Koch, S. (2007) – "Software Evolution in Open Source Projects – A Large-Scale Investigation", *Journal of Software Maintenance and Evolution*

2.2 Characterization of the Evolution Dynamics of Software

- **Expected Outputs/Results**
  Software evolution still lacks a theoretical model that explains why and how software evolves. Some studies have proposed that evolution dynamics is a self-organized criticality (SOC) dynamics. This study tests whether or not that dynamics model verify for the projects stored in FLOSSMetrics.

- **Data sources used**

Souce Code Management Systems	Yes
Mailing Lists	No
Issue Tracking Systems	No
Other (specify)	None

- **Methodology**
  For every project, the daily time series of number of changes was obtained. That time series was fitted to a particular model, to find out if the evolution was a short or long memory processes. If it is a short memory dynamics, its evolution dynamics cannot be SOC.

- **Filtering and other techniques applied on the data**
  - Kernel smoothing for the time series.

- **References**
  - I. Herraiz – "A statistical examination of the evolution and properties of libre software". *PhD thesis*, Universidad Rey Juan Carlos, 2008. http://purl.org/net/who/iht/phd.

  - I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles – "Determinism and evolution". *Proceedings of the 5th International Working Conference on Mining Software Repositories (MSR)*, pages 1–10. ACM, 2008.

2.3 Evolution of Core Team Members

- **Expected Outputs/Results**
  This study shows a methodology intended to characterize the evolution of the core teams of libre software projects. The methodology is based on the data available in the SCM databases of FLOSSMetrics. The results shown for the case of Evince show that in that project underwent a generational relay, and those top contributors at the beginning of the project are not contributing any more. This relay was very lively in the past history of the project, and it seems to have stopped in the recent history of the project, where the top contributors seem to have been present since some time ago.

- **Data sources used**

Souce Code Management Systems	Yes
Mailing Lists	No
Issue Tracking Systems	No
Other (specify)	None

- **Methodology**
  The methodology is based on dividing the history of the project in several periods of time, identifying the top committers within each period, and tracking the history of those top commiters in the rest of periods.

- **References**
  This work is based on a paper that has been submitted to the 6th International Working Conference on Mining Software: "*Evolution of the core team of developers in libre software projects".*

2.4 Evolution and Dynamics of Bugs

- **Expected Outputs/Results**
  - To understand the bug fixing and reporting process in open source projects.
  - To know how many bugs are reported and the average time it takes to fix a bug in a project's lifetime.
  - Find if there is any significant correlation between the size of the codebase and the number of bugs being reported.

- **Data sources used**

Souce Code Management Systems	Yes

Mailing Lists	No
Issue Tracking Systems	Yes
Other (specify)	None

- **Methodology**
  The methodology in this study can be summarized as follows:
  - Obtain data dumps from the FLOSSMetrics database.
  - Query the dumps for projects with both bug tracking and SVN data. Not all the projects with BTS have corresponding SVN data.
  - The resulted set of projects are used to extract some metrics for subsequent analysis.

- **References**
  - Gregorio Robles, Stefan Koch, Jesús M. González-Barahona – "Remote analysis and measurement of libre software systems by means of the CVSAnalY tool". http://libresoft.dat.escet.urjc.es/html/downloads/cvsanaly-icse.pdf
  - J. Anvik and C. N. Gail – "Determining implementation expertise from bug reports". Pages 1–8, 2007.
  - K. Ayari, P. Meshkinfam, G. Antoniol, and M. Di Penta – "Threats on building models from cvs and bugzilla repositories: the mozilla case study". In *CASCON '07: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, pages 215–228, New York, NY, USA, 2007. ACM
  - P. D. Christian Bird, Alex Gourley – "Detecting patch submission and aceptance in OSS projects", 2007.
  - C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller – "How long will it take to fix this bug" Page 1, 2007.
  - Chiara Francalanci and Francesco Merlo (Eds) – "Empirical Analysis of the Bug Fixing Process in Open Source Projects". Open Source Development, Communities and Quality, *IFIP International Federation for Information Processing*, Volume 275/2008, Springer Boston, 2008.
  - Sulayman K. Sowe, Ioannis Samoladas, Ioannis Stamelos, Lefteris Angelis (2008) – "Are FLOSS developers committing to CVS/SVN as much as they are talking in mailing lists Challenges for Integrating data from Multiple Repositories". *3rd International Workshop on Public Data about Software Development (WoPDaSD)*. September 7th - 10th 2008, Milan, Italy.

2.5 Effort

- **Expected Outputs/Results**
  The comparison between data from FLOSS projects and one version of the COCOMO model for proprietary software suggests that FLOSS productivity and effort estimation models are needed. The model presented here gives a first step to approach this research question and better models may be obtained refining and improving the measurement methodology. In this case, the model developed may be used to measure the possible impact of new or improved processes, methods and tools.

- **Data sources used**

Souce Code Management Systems	Yes
Mailing Lists	No
Issue Tracking Systems	No
Other (specify)	Source code releases

- **Methodology**
  The source code size is given by the number of lines which is retrieved from the table Metrics. Regarding the other three main measurements (effort, duration and team size) is provided by the CVSAnalY tool. The team size is given by number of committers who at least committed once. That number of people monthly ordered by their activity provide the effort in man-month and finally the duration is provided by the difference between the first commit and last commit.

- **References**
  This work is based on a paper accepted in the International Conference on Open Source Systems: Juan Fernandez-Ramil, Daniel Izquierdo-Cortazar, Tom Mens. “How Much Does It Take to Achieve One MegaLOC in Open Source”, *International Conference on Open Source Systems.*

2.6 Quality in Open Source Software

- **Expected Outputs/Results**
  First approach to measure how feasible is a community.

- **Data sources used**

Souce Code Management Systems	Yes
Mailing Lists	Yes

Issue Tracking Systems	Yes
Other (specify)	Source code releases

- **Methodology**
  For achieving its aims, QualOSS started by applying a GQM methodology, from which relevant goals, questions, and finally metrics and in dicators were derived. The computation of metrics measurements and aggregation for answering questions of the QualOSS quality model was partially automated with several tools.

- **References**
  This work is based on the research of QUALOSS project. A paper with these results was accepted in the International Conference on Open Source Systems (2009) : Daniel Izquierdo-Cortazar, Gregorio Robles, Jesus M. Gonzalez-Barahona, and Jean-Christophe Deprez – "Assessing FLOSS Communities. An Experience Report from the QualOSS Project", *International Conference on Open Source Systems*. Skövde, Sweden , 3-6 June 2009

2.7 Efficiency

- **Expected Outputs/Results**
  Because the relative efficiency measure is based on the distances from actually existing DMUs and is thus easily comprehensible, DEA is suitable as an analysis tool better than other methods. For the FLOSSMetrics project, the following variables are prime candidates for input measures:

  - Number of committers
  - Number of bug reporters
  - Number of posters
  - Effort estimations if possible (maybe using cumulated active developers as an indicator)

  The following metrics might provide a starting list for output variables:

  - Size
  - Number of postings
  - Bugs (with an important question being whether these should be modeled as a "bad", like pollution in other contexts, or whether bug reports are actually positive)
  - Software quality metrics
  - Growth rates

- **Data sources used**

Souce Code Management Systems	Yes
Mailing Lists	Yes
Issue Tracking Systems	Yes
Other (specify)	None

- **Methodology**

  Data Envelopment Analysis (DEA) is a non-parametric optimization method for efficiency comparisons without any need for the user to define any relations between different factors or a production function. In addition, DEA can account for economies or dis-economies of scale, and is able to deal with multi-input, multi-output systems in which the factors have different scales. Efficiency and productivity in software development is most often denoted by the relation of an effort measure to an output measure, using either lines-of-code or, preferably due to independence from programming language, function points. While this approach can be problematic in an environment of commercial software development as well due to missing components especially of the output, there are additional problems in the context of F/OS development which point towards DEA as an appropriate method.

  The output of a project can be measured using several software metrics like most easily the number of LOC, files, checkins to the source code control system, postings, bug reports, characteristics of development speed (e.g. coefficients of a software evolution equation estimated) or even metrics for product attributes like McCabe's cyclomatic complexity or object-oriented metrics, e.g. the Chidamber-Kemerer suite.

- **Filtering and other techniques applied on the data**
  - For preliminary results, all projects with data from all three main sources (source code, mailing list and issue tracking) have been used. Due to the switch to CVSAnalY2, this leaves (March 2009) 36 projects.

- **References**
  - Koch, S. (2009) – "Exploring the Effects of SourceForge.net Coordination and Communication Tools on the Efficiency of Open Source Projects using Data Envelopment Analysis", *Empirical Software Engineering*.

  - Koch, S. (2008) – "Measuring the Efficiency of Free and Open Source Software Projects Using Data Envelopment Analysis", in: Sowe, S.K., Stamelos, I. and Samoladas, I. (eds.): *Emerging Free and Open Source Software Practices*, pp. 25-44, IGI Publishing, Hershey, PA.

  - Koch, S. (2007) – "Exploring the Effects of Coordination and Communication Tools on the Efficiency of Open Source Projects using Data Envelopment Analysis", *Open Source Development, Adoption and Innovation (IFIP Working Group 2.13 on Open*

Source Software), IFIP International Federation for Information Processing Series, Vol. 234, pp. 97-108, Springer Verlag, Limerick, Ireland.

2.8 Correlation Size and Complexity Metrics

- **Expected Outputs/Results**
  Size and complexity metrics are highly correlated, and simple metrics can describe a software product. Another output is that software size is distributed like a double Pareto distribution, and that has implications on the modeling of software evolution.

- **Data sources used**

Souce Code Management Systems	Yes
Mailing Lists	No
Issue Tracking Systems	No
Other (specify)	None

- **Methodology**
  Metrics were gathered directly from the databases, and correlated using source code files as independent units. Then a regression analysis was done using that sample. The probability distribution functions for the different metrics were also estimated, and then fitted to particular models like lognormal or power law.

- **Filtering and other techniques applied on the data**
  - Empty files were removed

- **References**
  - I. Herraiz. – “A statistical examination of the evolution and properties of libre software”. *PhD thesis*, Universidad Rey Juan Carlos, 2008. http://purl.org/net/who/iht/phd.

  - I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles. – “Towards a theoretical model for software growth”. *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR)*, pages 21–28. IEEE Computer Society, 2007.

2.9 Project survival

- **Expected Outputs/Results**
  Since the beginning of free/open source software development thousands of new projects have been initiated. Out of these projects, a small fraction are actually alive and

continuously developed, while the majority of them are abandoned. The potential outcome of this kind of study is to determine the probability of the time point at which someone can claim that a free/open source project is inactive and thus considered as abandoned.

- **Data sources used**

Souce Code Management Systems	Yes
Mailing Lists	No
Issue Tracking Systems	No
Other (specify)	None

- **Methodology**
  The approach for determining the critical point of project survivability will be based on survival analysis (duration analysis). The data needed in order to build the probabilistic model will be supplied by the FLOSSMetrics database with the appropriate scripts. The analysis of these data will be facilitated through the usage of statistical software such as SPSS and the FLOSS package R.

- **Filtering and other techniques applied on the data**
  - Commits per day/week/month.
  - Posts on various mailing lists per day/week/month.
  - Bugs closed per day/week/month.
  - SLOC contributions (added or deleted) per day/week/month.
  - Developer base evolution.
  - Qualitative data such as application domain, type of information system, primary language (see also template on substitution cost estimation)
  - Empty files were removed

- **References**
  - Panagiotis Sentas, Lefteris Angelis and Ioannis Stamelos – “A statistical framework for analyzing the duration of software projects”, *Empirical Software Engineering*, Springer, 13 (2), 147-184

  - Nicholas Evangelopoulos, Anna Sidorova, Stergios Fotopoulos, Indushobha Chengalur-Smith – “Determining Process Death Based on Censored Activity Data”, *Communications in Statistics - Simulation and Computation*, Taylor-Francis, 37 (8),1647 – 1662

2.10 Maintainability Assessment

- **Expected Outputs/Results**
  The purpose of this study is to define whether there exists a measurable critical point that determines the route of a project as a possibility. The study will also attempt to determine entrance thresholds for a FLOSS project, i.e. activity related rates (bugs/time, code/time etc) that signal the attraction of many users/developers and therefore the creation of a critical mass of project participants.

- **Data sources used**

Souce Code Management Systems	Yes
Mailing Lists	Yes
Issue Tracking Systems	Yes
Other (specify)	None

- **Methodology**
  In order to determine how quality is affected by developers involvement and vice versa, we shall try to correlate specific source code metrics (like Source Lines of Code, Complexity Metrics, etc) and/or Mailing List and Bug Tracking data with the number of developers. The approach for determining the critical point of project survivability is based on tools, such as threshold analysis, cut-off curves, step functions. Statistical analyses, such as regression analysis or survival analysis might be useful.

- **Filtering and other techniques applied on the data**
  - Commits per day/week/month.
  - Bugs opened per day/week/month.
  - Bugs closed per day/week/month.
  - SLOC contributions (added or deleted) per day/week/month.
  - Developer base evolution.
  - Maintainability index evolution per day/week/month.

- **References**
  - I. Stamelos, L. Angelis, A. Oikonomou, G. Bleris. – “Code Quality Analysis in Open-Source Software Development”, *Information Systems Journal*, *2nd Special Issue on Open-Source*, Blackwell Science, Vol 12 (1), pp 43-60, 2002

2.11 Development vs. Communication

- **Expected Outputs/Results**
  The results of this study presents a possible methodology, its advantages and disadvantages which can benefit future researchers using some aspects of the FM retrieval system's data dumps.

- **Data sources used**

Souce Code Management Systems	Yes
Mailing Lists	Yes
Issue Tracking Systems	No
Other (specify)	None

- **Methodology**
  The methodology employed investigates F/OSS developers activities in both source code and mailing lists repositories. It aims to overcome challenges associated with identifying the simultaneous occurrence of developers in SVN and mailing lists. This has to do with difficulty in making sure that the developer making SVN commits in a project is the same individual posting to the developer mailing list(s) of the same project.

- **Filtering and other techniques applied on the data**
  - Source Code Management Systems
    - Identified and categorized commits as deletions, additions and modifications
  - Mailing Lists
    - Removing messages with “Subscribe” and “Unsubscribe” header messages
    - Removing postings with empty header messages and messages with headers but no contents
    - Unmasking aliases in which we identified individuals using different email addresses or names

- **References**
  - Sulayman K. Sowe, Ioannis Samoladas, Ioannis Stamelos, Lefteris Angelis (2008). – “Are FLOSS developers committing to CVS/SVN as much as they are talking in mailing lists? Challenges for Integrating data from Multiple Repositories”. *3rd International Workshop on Public Data about Software Development (WoPDaSD)*. September 7th - 10th 2008, Milan, Italy.

2.12 Contributor activity

- **Expected Outputs/Results**
  - Discover the quantitative dependence of individual contributor productivity on time, both in total but also per contribution type (functional change in code, bug fix, bug report).
  - Determine the average time dependence for all contributors
  - Through appropriate statistical analysis, determine whether there are clusters (groups) of contributors with different productivity time dependence and provide time curves for each such group

- **Data sources used**

Souce Code Management Systems	Yes
Mailing Lists	No
Issue Tracking Systems	No
Other (specify)	None

- **Methodology**

  By further statistical analysis over several scores of FLOSS projects, we will look for possible clustering of FLOSS projects with respect to programmer productivity or the existence of clusters of programmers with similar productivity patterns in production volume as well as its variation in time.
  Finally, we will use the new data to accordingly modify the simulation model parameters and apply it to particular FLOSS projects with the aim to obtain the time evolution of total LOC produced. We will also attempt a comparison between results of simulation model runs with and without the new data in order to check the expected improvement in the model's ability to reproduce the evolution of a FLOSS project realistically.

- **References**
  - I. Antoniades, I. Samoladas, I. Stamelos, L. Angelis, G.L.Bleris, in Eds: S. Koch. – "Dynamical Simulation Models of the Open Source Development Process"**,** *Free/Open Source Software Development,* IGI Global, 2005, and in *Global Information Technologies: Concepts, Methodologies, Tools, and Applications*, F. B. Tan, IGI Global, 2008

  - I. Antoniades, I. Stamelos, L. Angelis, G. Bleris. – "A Novel Simulation Model for the Development Process of Open-Source Software Projects" *Journal of Software Process Improvement*, Wiley, Special Issue on Process Simulation and Modeling, 7 (3-4) pp. 173-188, (2003)

2.13 Substitution Cost Estimation

- **Expected Outputs/Results**
  Substitution cost is the monetary value of the effort necessary for implementing a FLOSS application from scratch in a software company. This monetary amount has many potential uses, e.g. it may be used to estimate the gains from reusing a FLOSS component instead of building it from scratch. This pilot has helped identifying various problems, limitations and issues related to the data and the model precision. The ultimate target is the application of those models on the entire FLOSSMetrics code base.

- **Data sources used**

Souce Code Management Systems	Yes
Mailing Lists	No
Issue Tracking Systems	No
Other (specify)	ISBSG data, Qualitative information about FLOSS projects, such as business area type, language type

- **Methodology**
  The approach for estimating substitution cost is based on building multiple generic estimation models for the modern software industry and apply them collectively on whatever FLOSS code base we want to estimate. The latest version of ISBSG, a universal, publicly available, project cost database, is used to build estimation models. Up to now two estimation methods have been used, namely Regression and Analogy based. STATA tool has been used for regression and BRACE tool, enhanced with genetic algorithms, has been used for AbE. It is expected that other estimation approaches, such as machine learning algorithms (e.g. association rules) will provide further alternatives. The approach has been applied on a subset of projects from a Debian distribution, providing an approximation of the total substitution costs for this collection of FLOSS applications.

- **Filtering and other techniques applied on the data**
  - The major sources of data are ISBSG data and FM3 project data, namely SLOC and qualitative project characterisation. ISBSG data are filtered according to their quality, decided by maintainers of ISBSG: only A and B quality rating data are considered. SLOCs are retrieved from FM3 DB through the use of a script. Qualitative project data are not stored in the FM3 DB and therefore are retrieved independently.

- **References**
  - Kirsten Haaland, Ioannis Stamelos, Rishab Ghosh, Ruediger Glott. – “On the Approximation of the Substitution Costs for Free/Libre Open Source Software”, submitted to OSS2009

2.14 Productivity

- **Expected Outputs/Results**
  This study develops a spot productivity estimator function based on the survey data combined with the CVS data and applies this to the projects covered by the survey. Further extension to more projects is also possible.

- **Data sources used**

Souce Code Management Systems	Yes
Mailing Lists	No
Issue Tracking Systems	No
Other (specify)	Survey data

- **Methodology**
  The input data for the productivity study is a survey of committers to open source projects which was conducted in 2006. To participate in the survey the commiters had to make their CVS ID available, in order to unique identify an individual and relate it to the CVS tree of the corresponding project, available through FLOSSMetrics. This enables linking input and output and hence making the productivity estimator.

- **Filtering and other techniques applied on the data**
  - There were 240 respondents to the survey, but the final set being used for the analysis will probably be a bit lower due to cleaning and missing data. The availability of survey data dictates which CVS data will be included.

3. Focused Studies – Methodologies and Results

3.1 On the Validity of The Laws of Software Evolution

3.1.1 Introduction

The study of software evolution for commercial systems was pioneered by the work of Lehman and Belady on the releases of the IBM OS/360 operating system (Belady & Lehman, 1976), which has led to many other works, also based on other software systems (Lehman & Belady, 1985; Lehman & Ramil, 2001), in which the laws of software evolution were formulated, expanded and revised. These laws entail a continual need for adaptation of a system. This continual adaptation leads to increased complexity of the system. As it is assumed that constant incremental effort is applied at each time step, average incremental growth naturally declines. Turski (1996) for example has modeled this as an inverse square growth rate, others use a linear model. During the years, several other empirical studies have been conducted and published on software evolution in non-open source software systems, Kemerer & Slaughter (1999) and Scacchi (2004) provide overviews of such works.

For the area of open source software systems, several works have explicitly dealt with the topic of software evolution. The first and one of the most important contributions is a case study by Godfrey and Tu (2000), who have analyzed the most prominent example available, the Linux operating system kernel. Using the size in lines-of-code as a function of the time in days since release 1.0, they found that the growth behavior is best fitted by a super-linear rate. This result significantly contradicts the prior theory of software evolution, which postulates a decline in growth best modeled either using a linear or an inverse square rate. This would therefore give an indication of major differences in development models and their results.

On the other hand, Paulson et al. (2004) have used a linear approximation, and have not found any differences in growth behavior between open and closed-source software projects. In their study, they analyzed three widely known open source projects (Linux, Apache and GCC) and three closed-source systems.

Robles et al. (2005) reproduced the study of Godfrey and Tu (2000) with newer data, and found similar results, in addition showing that the growth of Linux has even accelerated during the five years between both works. They have also analyzed major subsystems, finding also super-linear growth patterns on this level. To validate these results, the authors have compared this to the family of *BSD kernels, which in contrast show an almost linear growth pattern with the exception of FreeBSD until the year 2000, where super-linear growth is present. In addition, 18 more large open source projects (like Apache, GNOME or KDE) were analyzed, finding growth patterns linear or close to linear for 16 of them, with the other 2 exhibiting some special characteristics. From the fact that the studied systems show a growth rate higher than a smooth one, the authors conclude that the fourth law of software evolution, ‘conservation of organizational stability’ which implies constant incremental effort, possibly does not apply to these large open source projects. In another case study, Robles et al. (2006) found that the KDE project shows super-linear growth.

Capiluppi et al. (2003) presented the first large horizontal study of open source system evolution using 406 projects from a repository, focusing more closely on 12 'alive' projects out of this set. In the full data set, the authors observe that over six months, 97% of projects did not change size or changed less than 1 percent. In the sample of alive projects, size is constantly growing, from which the applicability of the laws of software evolution is hypothesized, but no model is fitted to the data to further explore this notion. They also note that in large and medium projects, the number of modules grows, but their size tends to evolve to a stable value.

Robles-Martinez et al. (2003) have applied a similar methodology to this work based on the use of publicly available data in the form of the CVS (Fogel, 1999) source code control system, and have detailed the evolution of MONO as a case study from several perspectives, including commits, authorship and also size in lines-of-code. They conclude that the evolution of the studied modules proceeds with rather different growth rates, but did not fit any model to the data.

Nakakoji et al. (2002) have also studied the evolution of open source software systems, but take a broader perspective in also examining the evolution of the associated communities and the relationship between both types. Using four case studies, they link the system evolution on the level of different versions and branches to the community evolution, and arrive at a classification with three different types of projects.

Scacchi (2004) gives an excellent discussion of open source software evolution, with an overview and review of studies both on proprietary and open source projects. He concludes from this analysis that the laws of software evolution as presently stated and based primarily on the study of large closed source systems do not account for the potential for super-linear growth in software size that can be sustained by satisfied developer-user communities. He also stresses the importance of the availability of data on open source projects for further studies on software evolution and software engineering in general.

Koch (2007) studied the evolution of a large sample of several thousand open source software systems from Sourceforge.net. The evolutionary behaviour is explored using both a linear and a quadratic model, with the quadratic model being shown as better suited. The most interesting fact is that while in the mean the growth rate is linear or decreasing over time according to the laws of software evolution, a significant percentage of projects is able to sustain super-linear growth. There is a positive relationship between the size of a project, the number of participants and the inequality in the distribution of work within the development team with the presence of super-linear growth patterns. On the other hand, there is evidence for a group of projects of moderate size which shows decreasing growth rates, while small projects in general exhibit linear growth.

3.1.2 Methodology

Within FLOSSMetrics, we have replicated and refined this study. For computing and characterising the growth rates, the following methodology will be adopted. This is taken from a prior study of one of the project participants[1]. Note that the description pertains to growth in size (source code evolution), but is to be applied in similar terms for the other aspects of interest.

The first step is to analyse whether a linear or other growth pattern is present in the data. To this end, both a linear and a quadratic model are computed for each project, taking the size in

[1] Koch, S. (2007) "Software Evolution in Open Source Projects - A Large-Scale Investigation", *Journal of Software Maintenance and Evolution*, 19(6), pp. 361-382.

lines-of-code *S* as a function of the time in days since the first commit *t*, which is used as project start date, and using one month as time window. Therefore models A and B were formulated simply as:

$$S_A(t) = a * t + b$$
$$S_B(t) = a * t^2 + t * b + c.$$

The necessary parameters are to be estimated using regression techniques. As a next step, it is necessary to explore whether or not the growth rate is decreasing over time according to the laws of software evolution. This can be checked by analysing the second derivative of the quadratic model $S_B(t)'$, or more conveniently directly the coefficient of the quadratic term *a*.

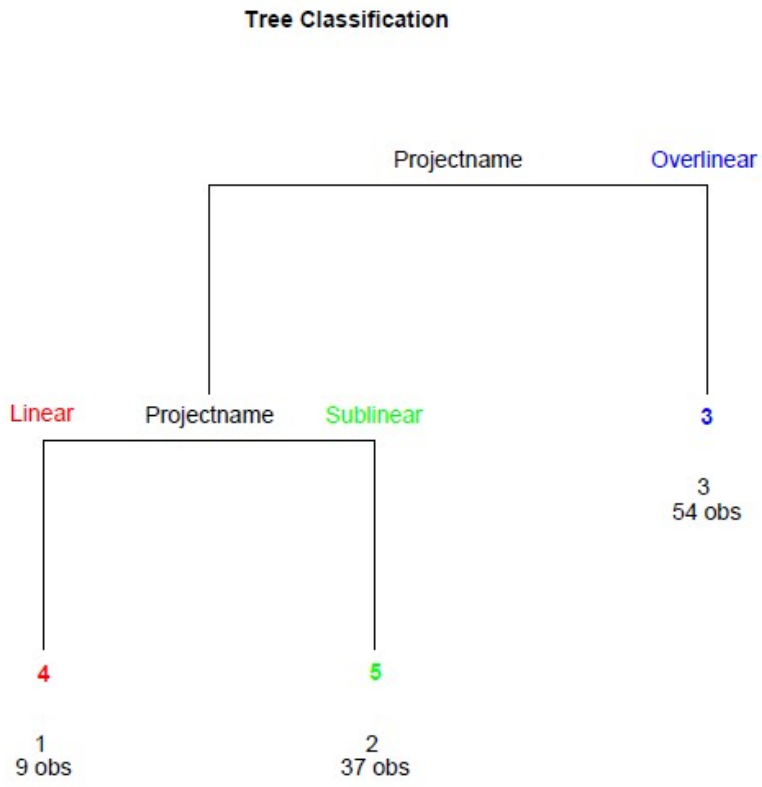
The sharp distinction between two groups of projects, those experiencing super-linear growth and those that don't, might prove to inflexible. A new group is therefore introduced representing linear growth in contrast to sub- and super-linear rates. This group is defined as those projects having either a better fit for the linear than the quadratic model, or a coefficient of the quadratic term between -0.1 and 0.1, thus being very near to zero. This allows for arriving, for each project and each aspect of interest, at a classification for the evolutionary behaviour as being either sub-linear, linear, or super-linear.

For characterising the past development, growth rates can be computed for several aspects. This should help to determine the speed in which the activity within the lifetime of the project has taken place. Growth rates can be computed for:

- Source code (constituting a study of software evolution)
- Programmers
- Bugs
- Bug reporters
- Mailings
- Posters

3.1.3 Results

The analysis was done using a set of 100 projects selected from the FLOSSMetric database. For each of those, the size in each month was retrieved and the evolution models were computed. For source code, this resulting in 9 projects being classified with linear growth patterns, 37 with sublinear growth patterns, and a majority of 54 as superlinear. This is in contrast to our prior study, which found linear rates were present in 42.3 percent, followed by 35.7 percent with sublinear and the smallest group of 22 percent with superlinear growth pattern. While the difference seems striking, the focus on large and active projects in the first phases of FLOSSMetric data retrieval can account for this: In our prior study, we found a relationship between project size and superlinear growth behaviour, which would explain why in this set of larger projects this type of behaviour is more common, and it actually underlines our findings.



The scripts for providing the model estimations and classification have been implemented using the R programming language. They will be included into the FLOSSMetric set of tools, and the classification according to growth behaviour is going to be included as a variable in the project database.

3.1.4 References

- Belady, L.A. & Lehman, M.M. (1976). – "A model of large program development". *IBM Systems Journal*, 15(3), 225-252.

- Capiluppi, A., Lago, P. & Morisio, M. (2003). – "Evidences in the evolution of OS projects through Changelog Analyses". *Proceedings of the 3rd Workshop on Open Source Software Engineering*, pp. 19-24, 25th International Conference on Software Engineering, Portland, Oregon.

- Godfrey, M.W. & Tu, Q. (2000) – "Evolution in Open Source software: A case study". *Proceedings. International Conference on Software Maintenance*, pp. 131-142.

- Hahsler, M. & Koch, S. (2005) – "Discussion of a Large-Scale Open Source Data Collection Methodology". *Proceedings of the Hawaii International Conference on System Sciences (HICSS-38)*, Big Island, Hawaii.

- Herraiz, I, Robles, G., González-Barahona, J.M., Capiluppi, A. & Ramil, J.F. (2006) – "Comparison between SLOCs and number of files as size metrics for software evolution analysis". *Proceedings 10th European Conference on Software Maintenance and Reengineering*, 2006.

- Kemerer, C.F. & Slaughter, S. (1999) – "An Empirical Approach to Studying Software Evolution". *IEEE Transactions on Software Engineering*, 25(4), 493-509.

- Koch, S. (2004) – "Profiling an open source project ecology and its programmers". *Electronic Markets*, 14(2), 77-88.

- Koch, S. & Schneider, G. (2002) – "Effort, Cooperation and Coordination in an Open Source Software Project: Gnome". *Information Systems Journal*, 12(1), 27-422002.

- Lehman, M.M. & Belady, L.A. (1985) – "Program Evolution – Processes of Software" Change. London: Academic Press.

- Lehman, M.M. & Ramil, J.F. (2001) - "Rules and Tools for Software Evolution Planning and Management". *Annals of Software Engineering*, 11, 15-44.

- Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K. & Ye, Y. (2002) - "Evolution Patterns of Open-Source Software Systems and Communities". *Proceedings of the International Workshop on Principles of Software Evolution*.

- Paulson, J.W., Succi, G. & Eberlein, A. (2004) – "An empirical study of open-source and closed-source software products". *IEEE Transactions on Software Engineering*, 30(4), 246-256.

- Robles, G., Amor, J.J., Gonzalez-Barahona, J.M. & Herraiz, I. (2005) – "Evolution and Growth in Large Libre Software Projects". *Proceedings International Workshop on Principles of Software Evolution (IWPSE 2005)*, Lisbon, Portugal.

- Robles, G., Gonzalez-Barahona, J.M. & Merelo, J.J. (2006) – "Beyond source code: The importance of other artefacts in software development (a case study)". *Journal of Systems and Software*, 79(9), 1233-1248.

- Robles, G., Koch, S. & González-Barahona, J.M. (2004) – "Remote analysis and measurement of libre software systems by means of the CVSAnalY tool". *Proceedings 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems,* 26th International Conference on Software Engineering, Edinburgh, Scotland.

- Robles-Martinez, G., Gonzalez-Barahona, J.M., Centeno-Gonzalez, J., Matellan-Olivera, V. & Rodero-Merino, L. (2003) – "Studying the evolution of libre software projects using

publicly available data". P*roceedings of the 3rd Workshop on Open Source Software Engineering*, pp. 111-116, 25th International Conference on Software Engineering, Portland, Oregon.

- Scacchi, W. (2004) – "Understanding Free/Open Source Software Evolution". In Madhavji, N.H., Lehman, M.M., Ramil, J.F. & Perry, D. (eds.), *Software Evolution*, New York: John Wiley and Sons Inc.

- Turski,W.M. (1996) – "Reference Model for Smooth Growth of Software Systems". *IEEE Transactions on Software Engineering,* 22(8), 599-600.

3.2 Characterization of the Evolution Dynamics of Software

Note: more information about this study can be found in the paper: "*Characterization of the evolution of software*" by Israel Herraiz. This paper is included as annexe to this document.

3.2.1 Introduction

Libre software development has been traditionally a source of strange cases of software evolution. The first to report one of those were Godfrey and Tu [6, 7]. Their findings suggested that the classical Lehman's laws of software evolution [15] were not fulfilled in the case of Linux, because it was evolving at a growing rate (which in fact was still growing 5 years later [20]). Those cases raised the question of whether libre software evolves differently than proprietary software, and whether the laws of software evolution are a valid approach for an universal theory of software evolution.

Although these questions have been addressed many times [14], most of the findings and models exposed on those works have failed to provide the theoretical background needed for a proper and universal theory of software evolution. One study that addressed the problem was Wu, in his PhD thesis [25], who among other interesting findings proposed that the evolution of libre software was governed by a Self Organized Criticality (SOC) dynamics.

This conclusion was supported by the presence of long range correlated time series in a set of 11 projects. Regardless the suitability of the selected projects for this kind of study, the limited amount of cases studies, or even the methodology used, we find the idea of long range correlated processes in software evolution as contrary to common intuition. Long range correlation would mean that the current state of the project is determined (or at least, heavily influenced) by events that took place long time ago. In other words, the evolution of libre software is governed by a sort of determinism.

In previous works [10, 8], it was already found that evolution was short range correlated for a sample of projects extracted from SF.net, contrarily to the results found by Wu. In this study, it will be repeated the methodology with a sample of projects extracted from FLOSSMetrics, to verify whether or not the same short range correlations hold for these cases. As in the mentioned papers, the daily time series of changes have been studied, focusing on deciding whether their profile were short or long range correlated.

3.2.2 Methodology

The methodology had three main steps:

1. **Data retrieval**
   Download the FLOSSMetrics aggregated database, and obtain all the data necessary for the analysis from the that database.

2. **Time series analysis**
   After gathering all the changes history, calculate the daily number of changes for each project. For each project, obtain the time series that will require further processing. It will be needed to apply a smoothing procedure to remove some noise from the data. After removing the noise, calculate the autocorrelation coefficients, in order to find out whether the project was a short or long term process.

3. **Statistical analysis**
   In order to quantify how many of the projects were described as short memory and how many as long memory, I used regression analysis and then analysed the distribution of these regressions. Some of these steps are detailed in the following subsections.

For a full description of the methodology of this study please visit the annexes of this document.

3.2.3 Results

In this study it has been considered the evolution of 78 software projects available in the FLOSSMetrics aggregated database. From that database, it has been used the activity in the version control system, in particular, the daily number of changes. Originally, the database contains 100 projects, but only 78 of them includes enough information to perform this study: either they are very young (younger than a year), or some statistical information like code size is missing).

For more information please visit the appendixes of this document.

3.2.4 References

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

- [2] G. Antoniol, G. Casazza, M. D. Penta, and E. Merlo. “Modeling clones evolution through time series”. In *Proceedings of the International Conference on Software Maintenance*, 2001.

- [3] 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.

- [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] 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”. J*ournal of Software Maintence: Research and Practice*, 12:343–355, 2000.

- [6] M. Godfrey and Q. Tu. “Evolution in Open Source software: A case study”. *In Proceedings of the International Conference on Software Maintenance*, pages 131–142, San Jose, California, 2000.

- [7] M. Godfrey and Q. Tu. “Growth, evolution, and structural change in open source software”. In *Internation al Workshop on Principles of Software Evolution*, Vienna, Austria, September 2001.

- [8] I. Herraiz. “A statistical examination of the evolution and properties of libre software”. *PhD thesis*, Universidad Rey Juan Carlos, 2008. http://purl.org/net/who/iht/phd.

- [9] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles. “Forecasting the number of changes in Eclipse using time series analysis”. In *International Workshop on Mining Software Repositories*. IEEE Computer Society, 2007.

- [10] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles. “Determinism and evolution”. In *Proceedings of the 5th International Working Conference on Mining Software Repositories* (MSR), pages 1–10. ACM, 2008.

- [11] I. Herraiz, J. M. Gonzalez-Barahona, G. Robles, and D. M. German. “On the prediction of the evolution of libre software projects”. In I*EEE International Conference on Software Maintenance*, pages 405–414. IEEE Computer Society, 2007.

- [12] I. Herraiz, D. Izquierdo-Cortazar, F. Rivas-Hernandez, J. M. Gonzalez-Barahona, G. Robles, S. Dueñas Dominguez, C. Garcia-Campos, J. F. Gato, and L. Tovar. “FLOSSMetrics: Free / libre / open source software metrics”. In *Proceedings of the 13th European Conference on Software Maintenance and Reengineering* (CSMR). IEEE Computer Society, 2009.

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

- [14] S. Koch. “Evolution of Open Source Software systems - a large-scale investigation”. In *Proceedings of the 1st International Conference on Open Source Systems*, Genova, Italy, July 2005.

- [15] M. M. Lehman and L. A. Belady, editors. Program Evolution. Processes of Software Change. Academic Press Inc., 1985.
- [16] M. M. Lehman, J. F. Ramil, and U. Sandler. ”An approach to modelling long-term growth trends in software systems”. In *International Conference on Software Maintenance*, pages 219–228, Florence, Italy, November 2001.
- [17] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski. “Metrics and laws of software evolution - the nineties view”. In *METRICS ’97: Proceedings of the 4th International Symposium on Software Metrics*, page 20, nov 1997.
- [18] N. H. Madhavji, J. Fernandez-Ramil, and D. E. Perry, editors. “Software Evolution and Feedback. Theory and Practice”. Wiley, 2006.
- [19] Y. Peng, F. Li, and A. Mili. “Modeling the evolution of operating systems: An empirical study”. *The Journal of Systems and Software*, 80(1):1–15, 2007.
- [20] 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.
- [21] 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.
- [22] R. H. Shumway and D. S. Stoffer. “Time Series Analysis and Applications. With R Examples”. Springer Texts in Statistics. Springer, 2006.
- [23] W. M. Turski. “Reference model for smooth growth of software systems”. *IEEE Transactions on Software Engineering*, 22(8):599–600, 1996.
- [24] W. M. Turski. “The reference model for smooth growth of software systems revisited”. *IEEE Transactions on Software Engineering*, 28(8):814–815, 2002.
- [25] J. Wu. “Open Source Software evolution and its dynamics”. *PhD thesis*, University of Waterloo, 2006.
- [26] J. Wu, R. Holt, and A. E. Hassan. “Empirical evidence for SOC dynamics in software evolution”. In *IEEE International Conference on Software Maintenance,* pages 244–254. IEEE Computer Society, 2007.
- [27] 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.

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

- [29] 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.

3.3 Evolution of Core Team members

Note: This work is based on a paper that has been submitted to the 6th International Working Conference on Mining Software: "*Evolution of the core team of developers in libre software projects*". This paper is included as annexe to this document.

3.3.1 Introduction

In many libre (free, open source) software projects, most of the development is performed by a relatively small number of persons, the "core team". The stability and permanence of this group of most active developers is of great importance for the evolution and sustainability of the project.

This study shows a quantitative methodology to analyse the evolution of core teams by retrieving information from source code management repositories. The most active developers in different periods are identified, and their activity is calculated over time, looking for core team evolution patterns. These date are represented in 3D graphs, that provides an overall picture about the evolution of core team members in a software project.

Employee turnover is known to be high in the traditional software industry since many years ago [1]. However, in libre software projects the study of developer turnover has not been an active research topic. Most of the attention in this area has been focused on the organizational structure of the projects, with little attention to the dynamics of the developers.

A noteworthy contribution in this sense, although it does not address the evolution of developer communities, is the o*nion model* [3], which shows how developers and users are positioned in communities. In this model, it is possible to differentiate among core developers (those who have a high involvement in the project), co-developers (with specific but frequent contributions), active users (contributing only occasionally) and passive users [4], [5].

The onion model provides only a static picture of a project, lacking the time dimension that is required for studying joining and leaving processes. Jensen *et al.* [6] have studied and modeled the processes of role migration for some libre software communities, focusing on end-users who become developers. This has lead to the finding of different paths for the joining process, concluding that the organizational structure of the studied projects is highly dynamic in comparison to traditional software development organizations. With respect to abandonment, the number of developers leaving a project has been studied in [7] by using the half-life parameter, defined as the time required for a certain group of contributors to fall to half of its initial population; in the case of the Debian project the obtained half-life was of 7.5 years.

This study shows a specific methodology to quantitatively characterize and visualize the evolution of the core team of the Evince project, the official document viewer for the GNOME desktop.

3.3.2 Methodology

The methodology is based on activity metrics obtained from the SCM databases of FLOSSMetrics. To characterize the evolution of the core team, first the life of the project is split in periods of equal duration. Then for every period *i*, the most active developers are identified as *Core Team i* . This is done by calculating the number of commits during that period for the most active developers. For each *Core Team i* , its activity is tracked for the rest of the life of the project (before and after period *i*). Hence, for each period *j*, the number of commits is calculated for all the developers in *Core Team i*. Finally, the resulting data (that represents the activity of the each *Core Team i* for all periods) is plotted that visualize the evolution of the core team group over the lifetime of the project.

In all the cases, only commits on source code files have been considered, since the study is focused on the behaviour of developers working on source code.

There are some cases which can be a source of problems when applying the methodology: both automatic committers (present in some projects for routine operations) and different patterns of work may influence the results for specific projects. However, experience after analyzing a large quantity of projects tends to show that these effects can, in general, be neglected. In addition, *Core Team i* and *Core Team j* may include developers in common, as a developer can be a part of the most active group in several periods. This is important to notice to correctly interpret the information provided by the methodology.

The selection of the duration of the periods in which the life of the project is divided is basic for the definition of the notion of “history” of a core team. The smaller the period, the more transient effects can be found (such as developers on vacation, or different activity patterns when the project is close to a release). But the larger the period, the lesser detail is captured, maybe losing important transitions, meaningful for this study.

In addition, projects of different life lengths will have different number of periods, if periods of constant duration are used. This could cause difficulties when comparing results, but will at the same time allow to better understand the behaviours related to the length of the history of projects. The opposite option, using a constant number of periods for projects of any length, can be more useful for comparisons, but will neglect those aspects related to project age.

Because of all these trade-offs, this study has not considered a single time span for periods. For the purposes of the study, usually the most significant results are obtained by dividing the history of the project into 10 or 20 periods. These also produce periods of a reasonable duration both for relatively young (about 2 to 4 months for projects with 3 years of development) and old projects (6 to 12 months for projects with around 10 years). To ensure that we are not losing any important information, we have obtained several plots using equal time intervals of three months.

Libre software projects usually show power law or similar distributions for the number of contributions by developers [4]. In other words, a small fraction of all the developers are responsible for a large fraction of all the activity. Because of that, the criterion chosen for the identification of core teams we have chosen is the fraction of developers who produce more commits. After considering several alternatives, we have found that fractions of 0.1 and 0.2 (that is, the top 10% and 20%) are large enough to capture developers producing most of the activity (usually more than 50%, reaching in many cases as much as 90% or 95% of the total number of commits).

Therefore, to summarize, we will track the evolution of the core team members (formed by the top 10% or 20% of committers), over 10 or 20 periods of time, and over periods of time of three months. The following queries will obtain the needed data. The first step is to create a table with the different kinds of period (20 periods, three months periods and so forth), that will be used later to track the activity of developers:

```
CREATE TABLE period_name
 id integer auto_increment,
 period integer,
 committer_id integer,
 n_commits integer,
 PRIMARY KEY (id)
)
```

Where PERIOD_NAME is a label for each kind of period. This table will be fed with the data about commiters and numbers of commit per period using the data that is available in the table scmlog. For this purpose, the following query can be used:

```
SELECT (to_days(s.date) - to_days(FIRST_DATE))
        div ((to_days(LAST_DATE) - to_days(FIRST_DATE))
        div NUM_SLOTS) period, s.committer_id,
count(distinct(s.id))
FROM scmlog s, actions a, file_types ft
WHERE a.commit_id = s.id AND a.file_id = ft.file_id AND ft.type
IN ('unknown', 'code')
GROUP BY period, s.committer_id
ORDER BY period, count(s.date) desc;
```

For this query, we need to specify the starting date of the study (FIRST_DATE), the ending data of the study (LAST_DATE), and the number of slots NUM_SLOTS. The dates can be determined using the first and last commit in the scmlog table. For the number of slots, if we are in the case of 10 or 20 periods, the number will obviously be 10 or 20. If it is three months, the number can be calculated with the first and last dates.

For each one commiter and period obtained from that query, the information can be piped to the PERIOD_NAME table. Thus, if for instance we study 20 periods and three months periods, we will have to PERIOD_NAME tables, one called 20 PERIODS (or similar) and the other one 3 MONTH (or similar). The above query has to be executed once for each kind of period.

Once the table PERIOD_NAME is filled, the top committers can be selected (for instance, the top 0.1 or 0.2 fractions, as explained above), and the data represented in a 3D graph, like those shown in the next section.

3.3.3 Results

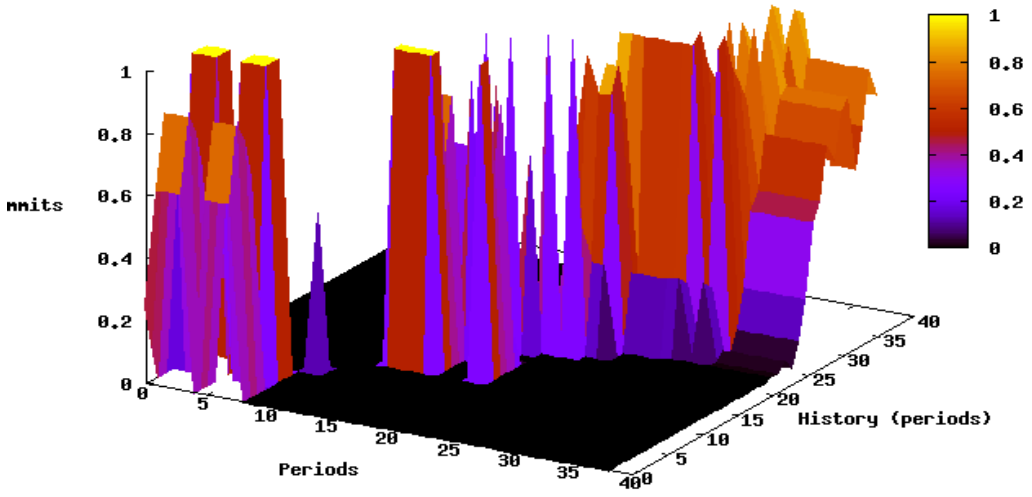
For the analysis of the case of Evince, the following combinations will be considered:

- Top 20% (fraction of 0.2) committers with periods of time of three months

- Top 20% (fraction 0.2) committers with 18 periods

These cases have been selected because the data was already available with those parameters, coming from previous works of some of the members of the URJC team.

The first case (0.2, three months) is shown in the following figure:

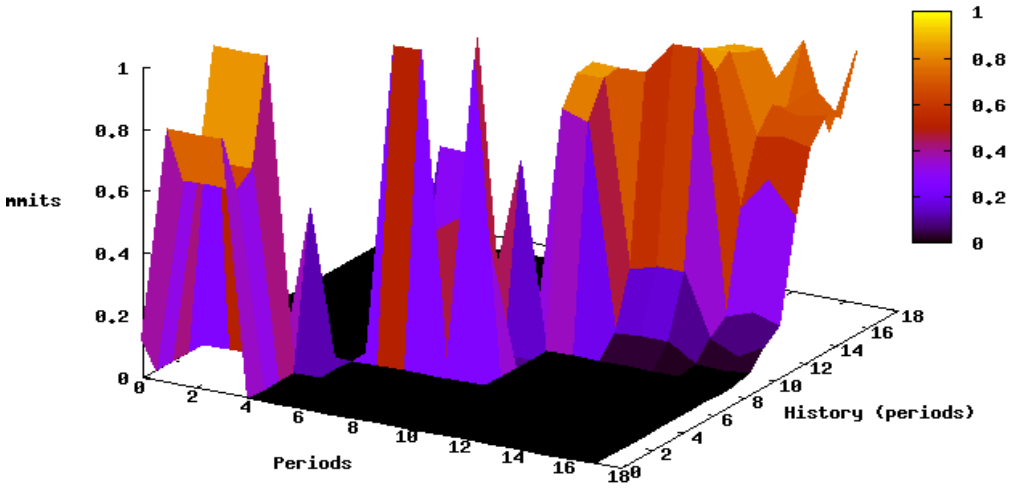


The vertical axis is normalized commits. The horizontal axis on the right side labeled as History (periods) shows each period of time. In the other horizontal axis, the core team in each period of time is shown. That core team is selected as the top 20% committers for each period, and then the history of that core team is tracked in the rest of periods. The value shown in the plot is normalized against the maximum number of commits.

For instance, if we focus in the case of the group 5, in the other horizontal axis (period 5), that core team has a value of 1, because they were the top commiters within that period. If we go now to the period 15 (without changing the value in the other axis), we observe that the value is null. That means that the core team #5 did not make any commit in that period. Therefore, the top commiters in period 5 were not contributing in period 15.

Interestingly the rest of core teams seem to have the same behaviour, except for the last 10 or 15 core teams. For instance, if we focus on the core teams 30 to 35, we observe that they started to contribute around period 25, and keep contributing in period 40. Therefore, it seems that the loss of contributors have been stopped in Evince.

Let us now analyse the case of the top 20% commiters with 18 periods of time. The figure is shown below:



This time, we have 18 periods of team, and 18 core team members. Each core team is formed by the top 20% commiters found in each period. Again, the data is normalized against the maximum number of commits.

The graph is very similar to the previous one. As the number of periods and core teams is lower, the graph seems less noisy, and the same behaviour depicted above is clearly observed in this case. For instance, the core teams from 0 to 4 were contributing very actively in the periods from 0 to 4, but from period 5 onwards, there is no trace of contributions coming from those developers.

Again, if we focus in the recent history of the project (core teams 12 to 18), we see that they started to contribute around period 10, and keep contributing in the last periods. This indicates a higher fidelity to the project that in previous core teams.

3.3.4 Conclusions

This study presents a methodology that allows to study the evolution of the core team of libre software projects. The methodology is quantitative, and can be automated, only requiring that the development is performed using a source control management system, and that the researcher has access to the corresponding repository. It provides also a lot of insight on the evolution of the core teams, by showing visually (both in graphs and maps) the activity patterns of the developers forming the core team in each period of the life of a project. This information can be used to identify levels of smoothness in transitions, to detect break points in the evolution of the core team, to understand the differences in activity of the core team in different periods, or to estimate unevenness in the contributions of the most active developers when compared to the rest of them.

This study was focused on Evince, but results from a total of 1361 projects can also be found in the FLOSSMetrics database for future analysis.

3.3.5 References

- [1] B. W. Boehm, Ed., “Software risk management”**.** Piscataway, NJ, USA: IEEE Press, 1989.
- [2] D. M. German, “The GNOME project: a case study of open a source, global software development,” *Journal of Software Process: Improvement and Practice*, vol. 8, no. 4, pp. 201–215, 2004.
- [3] K. Crowston and J. Howison, “The social structure of free and open source software development,” *First Monday*, vol. 10, no. 2, February 2005.
- [4] 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*, vol. 11, no. 3, pp. 309–346, 2002.
- [5] T. T. Dinh-Trong and J. M. Bieman, “The FreeBSD project:A replication case study of Open Source development,” *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 481–494, June 2005.
- [6] C. Jensen and W. Scacchi, “Modeling recruitment and role migration processes in OSSD projects,” in *Proceedings of 6thInternational Workshop on Software Process Simulation and Modeling*, St. Louis, May 2005.
- [7] 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*, Genoa, Italy, July 2005, pp. 100–107.

3.4 Evolution and Dynamics of Bugs

3.4.1 Introduction

The bug reporting and fixing process is an important constituent which has not been fully investigated in open source. Researchers often collect bugs data on one or two, at most nine, projects to study bug fixing regimes, how this process impacts the usability of open source software, use the bug fixing process as a measure of the quality and reliability of open source software, compare the bug fixing process in proprietary and open source software, study bug fixing policies, etc. One reason why most studies focus on few projects is because collecting and combining bug tracking databases from different projects has always been a difficult area. Our main motivation in carrying out this study is the availability of bug tracking data of hundreds and thousands of projects in the FLOSSMetrics database which we can use to study the Evolution and Dynamics of Bugs in Open Source Software Projects at a larger scale. With this large sample we expect to uncover patterns and offer new interpretations of the bug fixing process in open source projects, which other small-scale studies were not able to do.

This high level study which according to the research literature, for the first time, looks at the bug reporting and fixing process using data from more than one or two open source projects. Open source software projects are hailed for the rate at which volunteers contribute and fix bugs. A dynamic bug reporting and fixing process is an indication that the software is evolving and many security holes may be fixed just in time for shipment or release. This process calls for a active community participation from both developers and user. However, data from a single repository (bug tracking systems alone) from one or two projects will must likely not give a comprehensive picture of the dynamics of the bug contribution and fixing process in open source projects. In this research, we will use all the projects with both bug tracking and SVN data on FLOSSMetrics database. We determine the activities of individuals who participated in the bug contribution and fixing process using various metrics.

As a longitudinal research study, we make repeated observations of each and every bug in our sample from the time it is submitted and by whom, how the bug is being categorized (as a bug, feature requests, a patch to known vulnerability, as a support request, or as a result of miss-classification), who is assigned the bug and who actually fixed the bug, how the severity or priority status changes overtime, etc. Parameters in the bug fixing process (e.g total number of bugs reported, fixed, as well as the time it takes to fix the bugs) are compared with SVN data to determine if there is any correlation between these parameters and the complexity of the codebase. In addition our longitudinal study hope to answer some questions associated with the quality of open source software and offer an insight on how best open source developers can manage their bug fixing time, for example:

- If the time it takes to fix a bug increases, what are the implications for security of open source software?

- If critical bugs are miss-classified as feature requests or support requests, does this mean that they will go unnoticed and not fixed until the software is released?
- Are the developers who are assigned bugs actually the ones who fix them? If so how much time do they spend on bug fixing only?

The objectives of this project are the following:

- Understand the bug fixing and reporting process in open source projects.
- Know how many bugs are reported and the average time it takes to fix a bug in a project's lifetime.
- Find if there is any significant correlation between the size of the codebase and the number of bugs being reported.

3.4.2 Methodology

The figure below shows the bug reporting and fixing process from the perspective of three communities in bug tracking systems, source code versioning systems, and mailing lists.

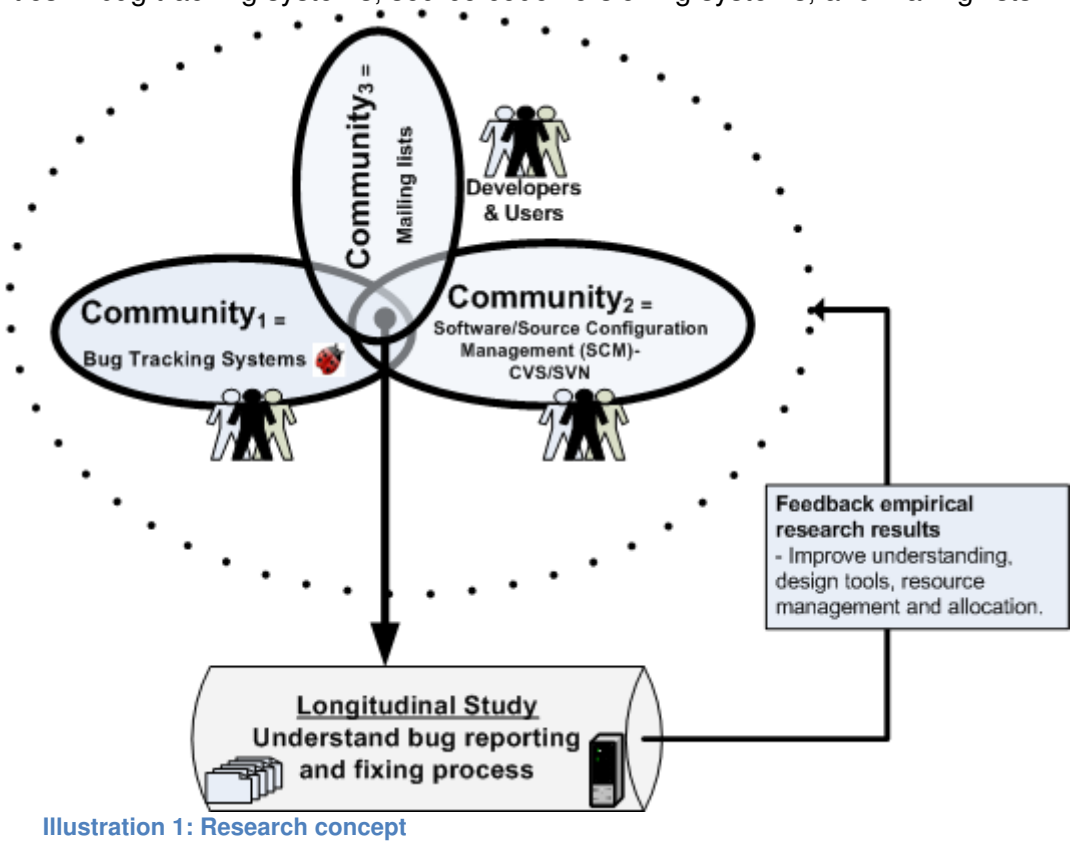


Illustration 1: Research concept

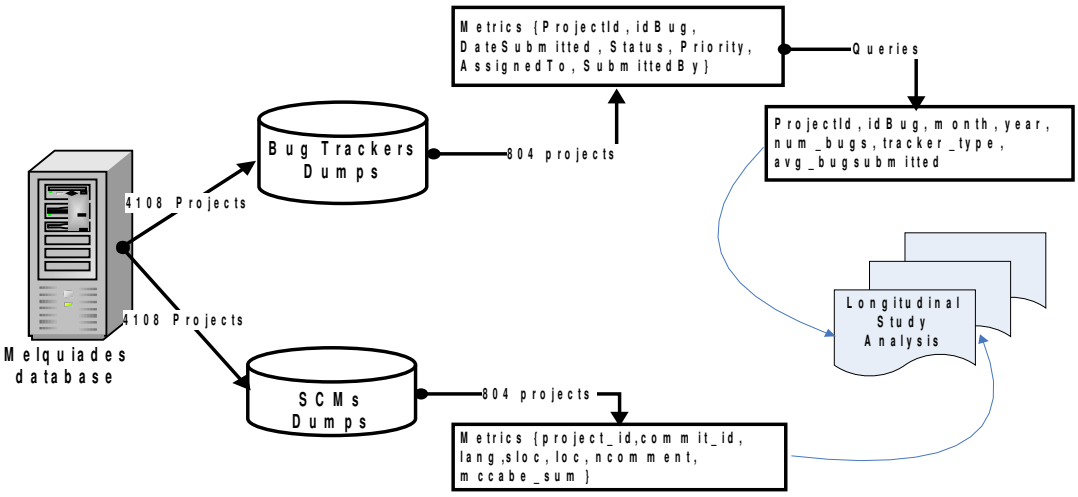
The methodology in this study can be summarized as follows:

- Obtain data dumps from the FLOSSMetrics database.

- Query the dumps for projects with both bug tracking and SVN data. Not all the projects with BTS have corresponding SVN data.
- The resulted set of projects are used to extract some metrics for subsequent analysis.

The entire methodology, excluding the data cleaning process, is schematically shown below.

Illustration 2: Methodology for studying the evolution and dynamics of bugs in FLOSS projects



The sources used in this study are the next:

- Bug tracking data from the *first level* (repository) of the FLOSSMetrics database. This was used to obtain metadata for our projects and bugs metrics.
- Aggregated data, *second level* of FLOSSMetrics databases, with data from Source Code Management Systems. This was used to obtain metadata on projects and SVN metrics.

Some queries used to extract the required metrics:

- *BugsxProject* query measures the number of bugs per project per monthly:

```
SELECT date_format(a.DateSubmitted, '%Y') myyear,
       date_format(a.DateSubmitted, '%M') mymonth,
       COUNT(a.idBug) num_bugs,
      (SELECT MIN (g.Tracker) FROM GeneralInfo g) Tracker
FROM Bugs a
GROUP BY myyear, mymonth
ORDER BY myyear, mymonth
```

- *SubmittersxProject* query measures the number bugs reporters per project per month:

```
SELECT year(b.DateSubmitted) myyear,
       date_format(b.DateSubmitted, '%M') mymonth,
       COUNT(DISTINCT b.SubmittedBy) num_submitter
FROM Bugs b
GROUP BY myyear, mymonth
ORDER BY myyear, mymonth
```

- *AvgBugSubmitter* query measures the number of average events per person participating in each data source (bug tracker, feature requests, patches, support requests, no-classified):

```
SELECT year(b.DateSubmitted) myyear,
       COUNT(b.idBug)/COUNT(DISTINCT b.SubmittedBy)
FROM Bugs b
GROUP BY year(b.DateSubmitted)
```

- *Delnotknown* query is an example of an analysis query which shows that 70.86% (2676) of the deleted bugs were assigned to nobody or anonymous persons in a project

```
SELECT idbug, datesubmitted, status,
       priority, assignedto, submittedby
FROM bugs
WHERE status='deleted' AND assignedto LIKE 'Nobody/anonymous'
```

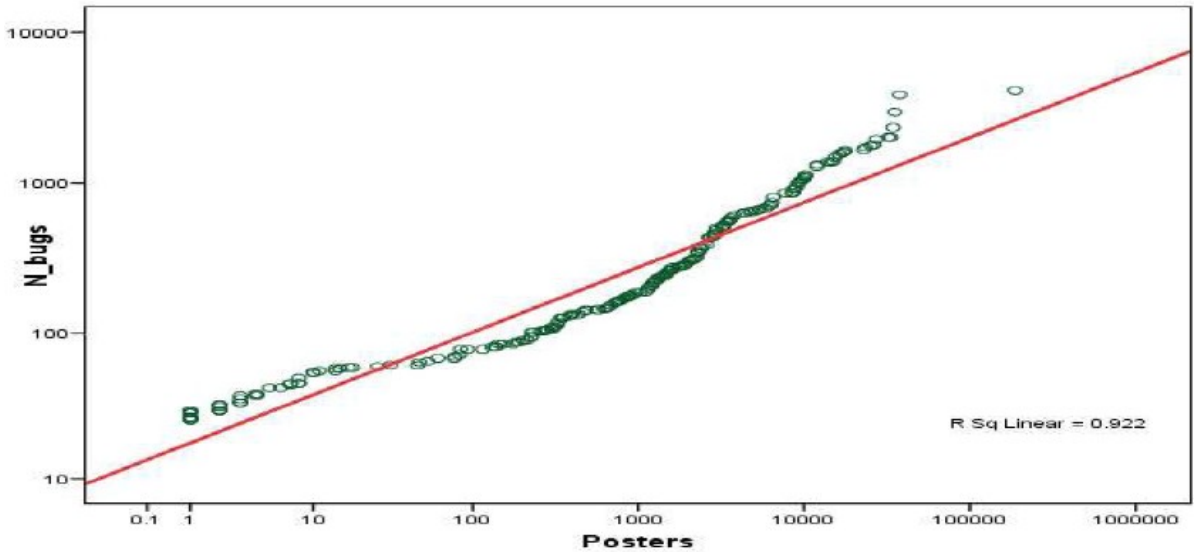
3.4.3 Results

Our results are aimed at answering three research questions.

1. ***What is the topology of bug communities like?***

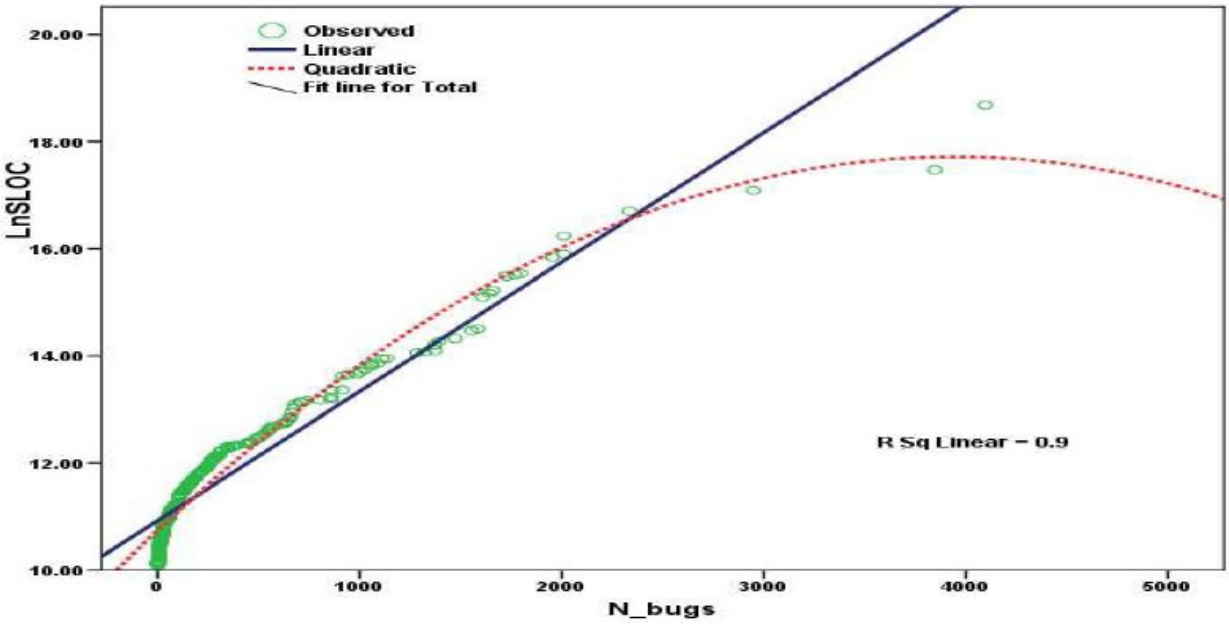
   A lot of effort is invested in helping improve the management, reporting, and resolution of bugs in F/OSS projects. Given that software is prone to bugs, perhaps we can reduce the bugginess and speed up the debugging process by understanding the way bug communities in various projects work. Bug reports usually have the ”identity” of the persons involved in bug triage; who submitted the bug, to whom the bug is assigned to, and who fixes the bug. In some instances, the assigned is the same person who fixes the bug. This identity is sometimes useful in helping software developers get in contact with bug reporters to confirm and discuss bug reports.
   To study the relationship between community size and the number of bugs reported, we counted the total number of posters in the mailing lists of all the project. We obtained 5448 posters. The mean and mode poster per project are, respectively, 19.12 and 3.00 (Std. Dev. = 86.598). The maximum number of posters in the largest project was 1259, minimum was 1. For each project we compared the total number of bugs reported (N bugs) with the total number of posters. Nonparametric correlations shows a significant relationship with = 0.790 ($p < 0.001$). The scatter plot in figure 7 shows the relationship fit with $R2 = .922$. As shown in table 6 the size of the communities are arranged in descending order of means.

The mean number of bug submitters in the 285 projects is 330.82 (Std. Dev. = 559.075). In 78.25% (N=223) of the projects non anonymous bug submitters (known people with names and emails) have their bugs assigned to non anonymous. The mean number of non anonymous bug submitters whose bugs are assigned to non anonymous individuals is 168.56 (Std. Dev. = 334.200). Furthermore, over 92% of those assigned bugs belong to members listed in the project's team. The mean number of anonymous bug submitters in 88.42% (N=252) of the projects is 92.97 (Std. Dev. = 208.896). This means that most bug submitters prefer to remain anonymous. However, in 80% (N=228) of the projects anonymous bug submitters have their bugs closed (mean= 70.41, Std. Dev. = 171.276). While in 31.92% (N=91) of the projects anonymous bug submitters (mean= 13.43, Std. Dev. = 29.753) had their bugs deleted. Furthermore, in 84.21% (N=240) of the projects, anonymous bug submitters have their bugs assigned to anonymous individuals in the projects' bug communities. The figure below summarizes the profiles of these communities in the 285 projects.



2. *Smaller projects in terms of SLOC have fewer bugs than large projects?*

In investigating the relationship between projects SLOC size and N bugs, we mapped each project's SLOC data with its corresponding bugs data. The outcome shows a significant correlation, with Pearson = 0.636, and Kendall's tau b = 0.998 for the ranked values; both correlations are significant at the 0.01 level (2-tailed). Furthermore, curve estimation regression statistics was applied to model the relationship between the two variables. As shown in the figure below, a linear model explains 90% (R2=0.900) of the variability, which is statistically significant. While quadratic model explains 95.2% (R2=0.952) of the variability.



Equations explaining this linearity for the original and transformed SLOC values and can be expressed as;

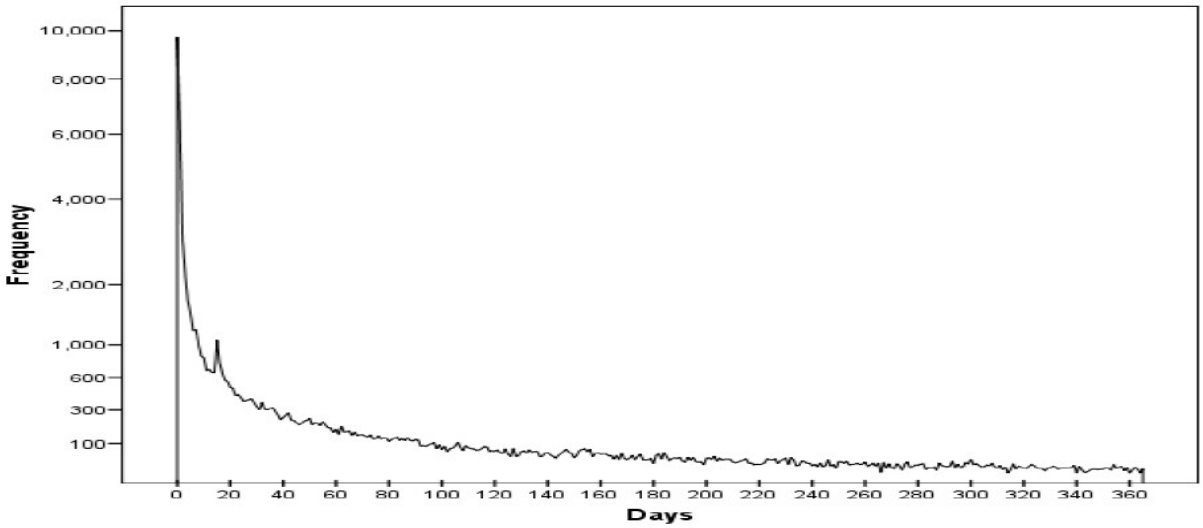
$$SLOC = 1.994094x10^{6} + -9.438X10^{3}(N\_bugs^{2})$$

$$lnSLOC = 10.92 + 2.4x10^{3}(N\_bugs)$$

3. ***What is the average time it takes for a bug to be closed?***

In our analysis we found out that the mean time to close a bug is 101 days (N = 68023, Mean = 17.00, Std. Deviation= 208.925). The maximum number of days it took to close the longest serving bug was 2270 days. The figure below shows the frequency of bugs closed in each day for the first 365 days. The peaks on the left of the chart shows that the majority of the bugs were closed within few days.

[2] *N_bugs* is the number of bugs in SLOC



TwoStep cluster analysis, using *timetoclose* or time to close a bug as a continuous variable and log-likelihood distance measure was applied to the *timetoclose* dataset. The auto-clustering algorithm revealed three main clusters. Bugs in cluster one represents 4.7% (N = 3209) of all the bugs and they take much longer time (from 535 to 2270 days) to close (mean = 868.04, Std. Dev. = 296.089). 15.7% (N =1 0656) of the bugs in cluster two were closed in about 2.5 times more than the average mean to close a bug. Cluster three consists of the majority, 79.6% (N=54158), of the bugs. The bugs in this cluster are closed below the mean time to close a bug, from 0 to 22.86 days. The importance of these attributes in each cluster can be compared using the mean value (101) as shown in figure 11.

3.4.4 Selected References

- Gregorio Robles, Stefan Koch, Jesús M. González-Barahona. “Remote analysis and measurement of libre software systems by means of the CVSAnalY tool”. http://libresoft.dat.escet.urjc.es/html/downloads/cvsanaly-icse.pdf

- J. Anvik and C. N. Gail. “Determining implementation expertise from bug reports”. Pages 1–8, 2007.

- K. Ayari, P. Meshkinfam, G. Antoniol, and M. Di Penta. “Threats on building models from cvs and bugzilla repositories: the mozilla case study”. In *CASCON ’07: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, pages 215–228, New York, NY, USA, 2007. ACM.

- P. D. Christian Bird, Alex Gourley. “Detecting patch submission and aceptance in OSS projects”, 2007.

- C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. “How long will it take to fix this bug” Page 1, 2007.

- Chiara Francalanci and Francesco Merlo (Eds). “Empirical Analysis of the Bug Fixing Process in Open Source Projects”. Open Source Development, Communities and Quality, *IFIP International Federation for Information Processing*, Volume 275/2008, Springer Boston, 2008.

- Sulayman K. Sowe, Ioannis Samoladas, Ioannis Stamelos, Lefteris Angelis (2008). “Are FLOSS developers committing to CVS/SVN as much as they are talking in mailing lists Challenges for Integrating data from Multiple Repositories”. *3rd International Workshop on Public Data about Software Development (WoPDaSD)*. September 7th - 10th 2008, Milan, Italy.

3.5 Effort

Note: This work is based on a paper accepted in the International Conference on Open Source Systems: "*How Much Does It Take to Achieve One MegaLOC in Open Source*". This paper can be found as appendix to this document.

3.5.1 Introduction

Effort modeling on open source systems is a key factor to better understand how it works and what kind of factors are involved. Libre software is mostly driven by volunteers, what means that projects are lead by people who, most of the times, work for free and in their spare time.

On the other hand, we find companies, whose employees are paid for a specific work and they always spend their time with activities regarding to the project. In this case, there is a big difference between these two ways to drive a project.

There are several papers whose main point is to measure effort in the proprietary world, however it does not exist in libre software and it is even more complicated to measure it due to that volunteer-driven projects.

This study tries to deepen in that field and to come to clarity some of the issues aforementioned.

3.5.2 Methodology

Run a query to the CVSAnalY databases for each system to determine the number of different person ids contributing to the source code management repository. The total person month was divided by 12 in order to get the man-year estimation.

Run a query to the CVSAnalY databases for each system to determine the number of files handled by month and see their evolution. In 9 projects out of 11, several jumps in the number of files were detected, what undertook a statistical process on the plot trying to remove them. It is unlikely those files were added just in one month. It is more probable that developers were working on their own and started a merge process.

Measure the total time duration as the difference between the first and last commit. Periods of inactivity were ignored.

Calculate the number of contributors. A contributor-month is calculated each time a commit is detected during a given month. In order to avoid people with really low activity, the core team was also calculated (those who have committed the 80% of the source code).

In order to retrieve specific data, the database had to be queried. Main metrics used in this paper are next:

- Number of physical lines of source code: This metric is given by SLOCCount
- Number of files in code repository: This metrics is provided by CVSAnalY
- Effort in contributor-years: This metric is the number of committers calculated per year.
- Time length of active evolution in years: This metrics is the project life measured in years.
- Number of distinct contributors: Unique contributors who have worked during the whole life of the project.

3.5.3 Results

In order to address our main research question which is how much it takes to develop 1 MLOC in FLOSS, we provide here some simple calculations, based on the best models identified in the previous section. Since our best model is based on file counts, we need to convert 1 MLOC into files. For the 11 FLOSS studied, the file size varied between 50 and 284 lines of code, with an overall average of 125 lines of code per file. Using this average8 , we can calculate that 1 MLOC will be equivalent to 8,000 files. Taking this value and using the linear model EFFORT vs netFILES excluding Eclipse, we obtain an estimate of 162.6 contributor years. Applying the model DUR vs netKLOC excluding Eclipse, we calculate an estimate duration of 8.6 years. Dividing 162.6 by 8.6 gives a core team size of 18.8 contributors.

Alternatively, if we use the model DEV vs netFILES excluding Eclipse, we obtain a team size of 41.9 developers. Dividing 162.6 by 41.9 gives an estimated duration of 3.8 years. The two values of DEV and DUR can be seen, respectively, as lower and upper bounds of the estimate. By finding some good theories, improving the extraction of the data suggested by these, adding more FLOSS systems to the sample and using more advanced techniques than linear regression, it may be possible to achieve more representative values.

The simple example illustrated how our very simple FLOSS-based models could be used. We are aware that, since FLOSS projects seek mainly to attract volunteers and not to hire paid professionals, the current practical use of productivity models will be very limited. In the future, however, measurements and models may find their way into helping FLOSS, probably in a yet unforeseen way.

3.5.4 References

- Juan Fernandez-Ramil, Daniel Izquierdo-Cortazar, Tom Mens. “How Much Does It Take to Achieve One MegaLOC in Open Source”, *Proceedings of Open Source Software Conference, 3-6 June, 2009, Skovde, Sweden.*

3.6 Quality in Open Source Software

This study is based on the QUALOSS results and on the paper: "*Assessing FLOSS Communities. An Experience Report from the QualOSS Project*", Daniel Izquierdo-Cortazar, Gregorio Robles, Jesus M. Gonzalez-Barahona, Jean-Christophe Deprez , *Proceedings of Open Source Software Conference*, 3-6 June, 2009, Skovde, Sweden. This paper can be found as appendix to this document. And a new version of the paper with new data and modified approach was sent to the CSMR 2010 edition. This is also a set of steps developed in the QualOSS project, which in the community side, aims to create an assessment model to deal with libre software communities.

3.6.1 Introduction

QUALOSS plans to mostly automate the quality measurement of open source software. The QUALOSS platform uses tools to analyse two types of data: source code and project-repository information. Thanks to the tooled-method of QUALOSS, it will be possible to assess the quality of open source projects quantitatively, objectively and rapidly. QUALOSS involves 8 partners from five countries, namely, Belgium, France, Germany, Spain and The Netherlands.

When acquiring software, enterprises are not only interested to know about the product and its quality but also interested in who produced that product and its reputability. For traditional enterprises, reputability can be check based on financial strength of the software provider however, for the FLOSS world, we must find other ways to determine if FLOSS endeavor (or FLOSS project) is serious. This can be done by studying the behavior of a FLOSS community. In particular, a FLOSS community should behaves in a manner to convince potential FLOSS integrators from Industry that it is dependable.

3.6.2 Methodology

The assessment process is divided in a series of 5 tasks: initiating an assessment, setting up and planning an assessment, collecting and analyzing data, interpreting results, and supervising an assessment.

1. **Initiating and Assessment**
   - Describe the broader context in which the assessment will take place, (e.g. the software development project that plans to integrate the FlOSS component).
   - State why an assessment is needed (in relation to a business viewpoint).
   - Explain how the outcome of an assessment will be used in the given broader context.

- Enumerate business constraints that the assessment activity must respect, e.g., limits on cost, timing and effort. These business constrains should derive from the first two objectives above.
- Validate the outcomes produced by the objectives above.

2. Setting up and planning an assessment

- Identify the accurate FlOSS endeavor(s) to assess, that is, the FlOSS endeavor's scope must be specified clearly and accurately
- Select the people who will perform the assessments and also identify the broader community of people likely to be impacted by the assessment (either because they will be contacted and asked to participate in the assessment activity, or because they may be impacted by the assessment results)
- Specify how people participating in the assessment will share the workload (using a workflow for example) and determine the rules that they will obey when performing the assessment.
- Identify the tools to use during the assessment activity and how these tools must be tailored or configured for this particular assessment. We note that in this context, the term tools is to be taken broadly, that is, tools may be automated (e.g. software tools), manual (e.g., manual procedures) or partially automated. For instance, manual procedures to follow to validate data or to evaluate documentation are considered tools and thus must be clearly identified. Naturally, software analysis tools should also be identified and included in the tools set to use later in the task of the assessment.
- Plan the supervision strategy for the remaining part of the assessment and communicated to the person in charge of the supervision task.
- Validate the outcomes produced by the objectives above

3. **Collecting and analyzing data**

- Collect the data whose sources were identified when setting up the assessment
- Analyze the data using the methods selected when setting up the assessment
- Maintain a link between the raw data, the methods and tools used to collect and analyze that data, the results of data analysis.
- Validate the collected and analyzed data

4. Interpreting the results

- Interpret the analyzed data resulting from the task collect and analyze data using the interpretation methods selected during the task of setting up and planning an assessment.
- Maintain a link between the analyzed data, the interpretation methods, and the resulting interpretation.
- Validate the resulting interpretation

5. Supervising and assessment

- Monitor that all the tasks of the assessment process are executed as planned including that each task performs the validation of its outcome.
- Record the tension that occurred during the whole assessment activity

In addition, this study had taken into account several points of view in the assessment process. Such as the role of product manager as the point of view to be used. In this way, this role has in mind a long term vision of the project. In fact, this role is more interested to know about the evolution trend of a FLOSS endeavour rather than to know about the current state of an FLOSS endeavour. From this point of view, and following the GQM approach, this paper defined several questions to be answered.

Next there is a brief example of those questions:

1. Has the evolution of new core contributing members remained stable or grown over the history of the FLOSS endeavor? (at least shown a stable or positive trend overall) (a core member is one with commit right whoperform commits frequently for instance, more than once every three month period)

2. Has the evolution of core members who stopped contributing for a significant period been compensated by the joining of new core members around the same time frame?

3. What is the average longevity of committers

To answer the set of questions and describe the corresponding community aspects of FLOSS endeavors, metrics were defined. Next, there is a set of them as an example:

sra7 average number of months where each committer committed,
iwa4 proportion of files maintained by a single committer,
sra2 number of new code committers in the interval,
sra3 number of new non code committers in the interval,

3.6.3 Results

This work presents the results of using the QualOSS method applied on libre software communities. The methodology is based on the analysis of around 1400 projects from the FLOSSMetrics repository. The data sources which were used are the source code management system, the mailing lists and the bug tracking system. However, in order to have better results, for this study, only the source code management system was analysed since it was the one with more data available.

Metrics defined as an example:

1 **Evolution_first_commit_submitted_by_registered_people**

1.1 *Rationale*: Retrieving the date of the first commit for each member of the community, we are able to know if the number of new member committing remains stable

1.2 *Query*

```
SELECT g.yearmonth, count(g.committer_id)
FROM (
      SELECT s.committer_id,
             date_format(MIN(s.date), '%Y%M') yearmonth
      FROM scmlog s, actions a, file_types ft
      WHERE s.id=a.commit_id AND a.commit_id=ft.commit_id
        AND a.file_id=ft.file_id and ft.type='code'
      GROUP BY s.committer_id
) g
GROUP BY g.yearmonth order by  g.yearmonth;
```

1.3 *Indicator definition:* Taking into account the slope of the resultant line (y=mx+b) while measuring the aggregated number and periods of one year:

- Black:]–∞, –3]
- Red [-2,-1]
- Yellow [0,1]
- Green: [2,+∞]

2 Territoriality

2.1 *Rationale:* This metric shows the territoriality in a project. Generally speaking, most of the files are touched or handled by just one committers. It means that high levels of orphaning may be seen as a risk situation. If a developer leaves the project, her knowledge will disappear and all her files are totally unknown by the rest of the developers team.

2.2 *Query*

```
SELECT (COUNT(g.file_id) / (
        SELECT COUNT(DISTINCT file_id)
        FROM actions))*100
FROM   (
        SELECT a.file_id,
               COUNT(DISTINCT s.committer_id)
        FROM actions a, scmlog s
        WHERE a.commit_id=s.id
        GROUP BY a.file_id
        HAVING COUNT (DISTINCT s.committer_id)=1
) g;
```

2.3 *Indicator definition*

- Black:]0.87245, +∞]
- Red:]0.7124, 0.87245]
- Yellow:]0.5522, 0.7124]
- Green: [0, 0.5522]

All in all, next there is a set of results and their evolution through time:

Next there is a comparison between thresholds defined in version 1.0 and version 1.1. If a "-" is found, it means that the thresholds are the same as the previous version. :

Metrics	***V1.0***	***Intermediate***	***V1.1***
cm-sra1	Black:]-∞, -5] Red:]-5,0[Yellow: 0 Green:]0,+∞[	-	-
cm-sra2	Black:]-∞, -5] Red:]-5,0[Yellow: 0 Green:]0,+∞[	Black:]-∞, -0.3167] Red:]-0.3167, -0.1469] Yellow:]-0.1469, -0.0527] Green:]-0.0527, +∞[	Black:]–∞, –3] Red: [–2, –1] Yellow: [0, 1] Green: [2, +∞]
cm-sra3	Black:]-∞, -5] Red:]-5,0[Yellow: 0 Green:]0,+∞[	Black:]-∞, -0.7] Red:]-0.7, -0.2647] Yellow:]-0.2647, -0.1135] Green:]-0.1135, +∞[	Black:]–∞, –3] Red: [–2, –1] Yellow: [0, 1] Green: [2, +∞]
cm-sra4	Black:]-∞, -5] Red:]-5,0[Yellow: 0 Green:]0,+∞[	Black:]–∞, –0.5] Red:]–0.5, –0.2] Yellow:]–0.2, –0.0357] Green:]–0.0357, +∞[	Black:]–∞, –2] Red: –1 Yellow: 0 Green: [1, +∞]
cm-sra5	Black: [3, +∞[Red:]1, 3] Yellow:]0, 1] Green: 0	Black:]0.4, +∞[Red:]0.1, 0.4] Yellow:]0, 0.1] Green:]–∞, 0]	Black: [1, +∞] Red: 0 Yellow: -1 Green:]–∞, –2]
cm-sra6	Black:]-∞, -5] Red:]-5,0[Yellow: 0 Green:]0,+∞[	Black:]–∞, –0.7636] Red:]–0.7636, –0.3] Yellow:]–0.3, –0.1429] Green:]–0.1429, +∞[	Black:]–∞, –2] Red: –1 Yellow: 0 Green: [1, +∞]
cm-sra7	Black: [0, 12[Red: [12, 24[Yellow: [24, 36[Green: [36, +∞[	Black: [1, 5.53555] Red:]5.53555, 8.5789] Yellow:]8.5789, 12.25] Green:]12.25, +∞]	-
cm-sra8	Black:]-∞, -5] Red:]-5,0[	-	-

	Yellow: 0 Green:]0,+∞[		
cm-sra9	Black:]-∞, -5] Red:]-5,0[Yellow: 0 Green:]0,+∞[	Black:]–∞, –0.2364] Red:]–0.2364, –0.0593] Yellow:]–0.0593, 0.0901] Green:]0.0901, +∞[	Black:]–∞, –3] Red: [–2, –1] Yellow: [0, 1] Green: [2, +∞]
cm-iwa1	Black:]-∞, -5] Red:]-5,0[Yellow: 0 Green:]0,+∞[	Black:]–∞, –13.4047] Red:]–13.4047, –1.5654] Yellow:]–1.5654, 4.1337] Green:]4.1337, +∞[	Black:]–∞, –39] Red:]–39, –1] Yellow:]–1, 23.7] Green:]23.7, +∞]
cm-iwa2	Black:]-∞, -5] Red:]-5,0[Yellow: 0 Green:]0,+∞[	Black:]–∞, –19.4] Red:]–19.4, –3.6696] Yellow:]–3.6696, 0.8735] Green:]0.8735, +∞[	Black:]–∞, –37.2] Red:]–37.2, –2] Yellow:]–2, 18.5] Green:]18.5, +∞]
cm-iwa3	Black: 0 Red:]0, 0.05[Yellow: [0.05, 0.1[Green: [0.1, 1]	-	-
cm-iwa4	Black: [0.9, 1] Red: [0.7, 0.9[Yellow: [0.5, 0.7[Green: [0, 0.5[	Black:]0.87245, +∞] Red:]0.7124, 0.87245] Yellow:]0.5522, 0.7124] Green: [0, 0.5522]	-
cm-iwa5	Black: [100000, +∞[Red: [50000,100000[Yellow: [30000, 50000[Green: [0, 30000[	Black: [0, 2130.7] ∪]72295, +∞] Red:]2130.7, 5293.5] ∪] 46029.1, 72295] ∪]29543.5, 46029.1] Yellow:]5293.5, 9791.7] Green:]9791.7, 29543.5]	-
cm-iwa6	Black: [100000, +∞[Red: [50000,100000[Yellow: [30000, 50000[Green: [0, 30000[	Black: [0, 2130.7] ∪]72295, +∞] Red:]2130.7, 5293.5] ∪] 46029.1, 72295] ∪]29543.5, 46029.1] Yellow:]5293.5, 9791.7] Green:]9791.7, 29543.5]	-
cm-iwa7	Black: [500, +∞[Red: [200,500[Yellow: [50,200[Green: [0,50[	Black: [0, 0.012875] ∪] 0.364425, 1] Red:]0.012875, 0.0443] ∪] 0.2555, 0.364425] Yellow:]0.0443, 0.076625] ∪]0.188075, 0.2555]	-

		Green:]0.076625, 0.188075]	

3.6.4 Conclusions

We have presented a way to estimate ”quality” from an industrial perspective based on statistical analysis of hundreds of FLOSS projects. FLOSS communities are key actors in the software evolution and maintenance process and better understanding their behavior through their life will improve the make decision process. For instance, checking the tendency by means of the methodology explained in this paper, we are able to know if a community is growing, or if the number of core committers is decreasing over and over. Taking into account the latter, we may not be interested in adding more effort in a given project due to the fact that main developers are leaving the project and this is a tendency checked during last months or years. On the other hand, perhaps we are interested in a specific product and we know that some of its weaknesses are motivated because of a really high turnover of developers. It is also important to check the status of potential activity per committers. Sometimes it is necessary, for maintenance purposes to check how many files are being handled by committers in order to check if they are overloaded.

3.6.5 References

- Daniel Izquierdo-Cortazar, Gregorio Robles, Jesus M. Gonzalez-Barahona, Jean-Christophe Deprez , “Assessing FLOSS Communities. An Experience Report from the QualOSS Project”. *Proceedings of Open Source Software Conference*, 3-6 June, 2009, Skovde, Sweden.

3.6.6 Conclusions using the FLOSSMetrics data

Around 1400 projects have been used to identify potential limits for the indicators defined in the paper. Some of them have been detected as minor projects since their communities are smaller than three developers in total. However this is a usual situation in statistics to make a previous filtering in the data what means that the FLOSSMetrics projects provides every kind of possible libre software projects. However for the study presented here we needed to remove some of them to achieve some of the goals.

3.7 Efficiency

3.7.1 Introduction

For any production process, this efficiency and productivity is a key indicator in comparison to other processes. To this end, we will apply the method of Data Envelopment Analysis (DEA), which is a non-parametric optimization method for efficiency comparisons without any need for the user to define any relations between different factors or a production function. In addition, DEA can account for economies or dis-economies of scale, and is able to deal with multi-input, multi-output systems in which the factors have different scales. Efficiency and productivity in software development is most often denoted by the relation of an effort measure to an output measure, using either lines-of-code or, preferably due to independence from programming language, function points. While this approach can be problematic in an environment of commercial software development as well due to missing components especially of the output, there are additional problems in the context of F/OS development which point towards DEA as an appropriate method.

In F/OS projects, normally the effort invested is unknown, and therefore might need to be estimated, and is also more diverse than in commercial projects, as it includes core team member, committers, bug reporters and several other groups with varying intensity of participation. Besides that, also the outputs can be more diverse. In the general case, the inputs of an F/OS project can encompass a set of metrics, especially concerned with the participants. So, in the simplest case, the number of programmers and other participants can be used. The output of a project can be measured using several software metrics like most easily the number of LOC, files, checkins to the source code control system, postings, bug reports, characteristics of development speed (e.g. coefficients of a software evolution equation estimated) or even metrics for product attributes like McCabe's cyclomatic complexity. This range of metrics both for inputs and outputs, and their different scales necessitates application of an appropriate method, which DEA can be.

The main result of applying DEA for a set of projects is an efficiency score for each project. This score can serve different purposes as mentioned above: First, single projects can be compared accordingly, but also groups of projects, for example those following similar process models, located in different application domains or simply of different scale can be compared to determine whether any of these characteristics lead to higher efficiency.

3.7.2 Methodology

The basic principle of DEA can be understood as a generalization of the normal efficiency evaluation as described above by means of the relationship from an output to an input into the general case of a multi-output, multi-input system without a any given conversion rates or same units for all factors. For the area of software development, this can be understood as adding for example pages of documentation to lines-of-code and function points as an output factor. In contrast to other approaches, which require the parametric specification of a production function, DEA measures production behavior directly and uses this data for the evaluation of all DMUs. The

DEA derives a production function from mean relations between inputs and outputs (whereby it is only assumed that the relation is monotonous and concave), by determining the outside cover of all production relations, while for example a regression analysis estimates a straight line through the center of all production relations. The DEA identifies "best practicing" DMUs, which lie on the production border. Thus any outliers and measuring and / or data errors may exert a strong influence on the results, and therefore special attention is to be given to sensitivity analyses. In order to determine the production function, the efficient DMUs are linked in sections with one another. A DMU is understood as being efficient if none of the outputs can be increased, without either or several of the inputs increasing or other outputs being reduced, as well as none of the inputs can be reduced, without reducing either one or more outputs or increasing other inputs.

For each DMU an individual weighting procedure is used over all inputs and outputs. These form a weighted positive linear combination, whereby the weights are specified in such a way that they maximize the production relationship of the examined unit, in order to let these become as efficient as possible. The efficiency of an examined unit is limited with 1. That means it that no a-priori weightings are made by the user, and that the weights between the DMUs can be different. However, the definition of the relevant inputs and outputs is necessary. For each evaluation object the DEA supplies a solution vector of weighting factors and a DEA efficiency score. If this score is equal to 1, then the DMU is DEA efficient. In this context, DEA efficiency means that within the selected model variant no weighting vector could be found which would have led to a higher efficiency value. DEA efficient are thus all those DMUs which are not clearly DEA inefficient compared with the others. Any inefficiency can therefore not be ruled out completely. For inefficient DMUs weighting factors were found in the context of the selected model variant which resulted in a higher efficiency value in the case of at least another DMU.

For each inefficient DMU the DEA returns a set of efficient DMUs which exhibit a similar input/output structure and lie on the production border near to the inefficient DMU (this is also termed reference set or DEA benchmark). Using this information, an idea in which direction an increase in efficiency is possible can be gained. Because the relative efficiency measure is based on the distances from actually existing DMUs and is thus easily comprehensible, DEA is suitable as an analysis tool better than other methods.

The different basic models of the DEA can be divided on the basis of two criteria: This is on the one hand the orientation of the model, on the other hand the underlying assumption regarding the returns to scale of the production process. With input-oriented models the reduction of the input vector maximally possible with the given manufacturing technology is determined, whereas with output-oriented models the maximally possible proportional increase of the output vector is determined. Input orientation is generally present, if an enterprise function (for example manufacturing) is to be analysed which minimizes the resources consumption, but can only affect the output in a limited way. The returns to scale can be assumed either as being constant as in the CCR model. With constant returns to scale size-induced productivity differences are considered into the efficiency evaluation, with variable returns to scale the differences are neutralized by the model. The most common example of a model with variable returns to scale is an advancement to the CCR model, the BCC model. This model includes an additional measuring variable in the fundamental equation to capture rising, falling or constant returns to scale.

The first choices to be taken for any application concern the definition of input and output factors, as well as the model to be applied. Regarding the orientation of the model, an output-orientation might seem more appropriate. Given a certain input which can be acquired, i.e. participants attracted, the output is to be maximized. Most studies in software development have found variable returns to scale. According to this reasoning, the BCC-O model is applied.

Regarding the definition which factors are to be used as inputs and outputs, it is to be considered that with an increase in the number of factors more DMUs, i.e. projects, are estimated to be efficient, in particular if the database is small in relation. Also the availability of factors in the data set limits the possibilities. For the FLOSSMetrics project, the following variables are prime candidates for input measures:

- Number of committers
- Number of bug reporters
- Number of posters
- Effort estimations if possible (maybe using cumulated active developers as an indicator)

The following metrics might provide a starting list for output variables:

- Size
- Number of postings
- Bugs (with an important question being whether these should be modeled as a “bad”, like pollution in other contexts, or whether bug reports are actually positive)
- Software quality metrics
- Growth rates

The DEA computations will be done in R, a statistical environment, using the DEA package (see references for full information).

3.7.3 Results

At the current date (March 2009), and due to the switch to CVSAnalY2, only a limited set of projects is available. As we tried to include all different input types (i.e. programmers, posters and mailers), we only have 36 projects left. Using queries to the FLOSSMetrics database, the different variables were retrieved from each of the three sources (source code control, issue tracking and mailing lists), and were merged based on project identification. This resulted in a file for the 36 projects, with the following variables included in this preliminary study:

- Number of committers
- Number of bug reporters
- Number of posters
- Number of commits
- Number of codefiles
- Total LOC

- Number of bugs
- Number of messages

Using this dataset, we are able to, for the first time, capture all different forms of inputs and outputs within an open source project. According to prior research, participants form a sort of onion model, with an inner core, programmers, and other active participants. In this efficiency calculation, we can now capture all of those different elements. As input factors, we therefore used the number of committers, bug reporters, and mailing list submitters. For output, we used in a first approach all other available measures, code-, bug- and mail-related. It might be discussed whether bugs constitute a positive output of a project, but for now, the line of reasoning that finding a bug is a good thing, and that some of these issue actually constitute additional requirements submitted, is adopted. Other possible output factors of interest would include some kind a measure for user acceptance, like number of downloads, installations or similar. This is not currently available.

Using this setup, and, as discussed above, using a BCC model, efficiency scores were computed using R and DEA library 0.1-2. A portion of the R source code for this is present here for illustration (please note that some of the additional analyses at the end are not being used at this point):

```
rm(list=ls())
library(DEA);
projects <- read.csv("daten.csv", header=TRUE)
attach(projects)
inputs <- matrix(nrow=length(number_developers),ncol=3)
inputs[,1] <- number_developers
inputs[,2] <- CountBugsPeople
inputs[,3] <- CountMsg_People
... (some code omitted)
erg <- dea.bcc.oo.mul(inputs,outputs)
eff <- 1/erg$eff
for (i in 1:length(eff)) {if (eff[i]==Inf) eff[i] <- 0}
projects$eff <- eff
summary(projects$eff)
sd(projects$eff)
```

The results show a very small variation in DEA efficiency, with actually a majority of projects being seen as DEA efficient. The following statistics give a descriptive overview of the efficiency scores:

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.9513	0.9998	1.0000	0.9961	1.0000	1.0000

One reason for this could be the currently high number of factors as compared to size of data set. In such a case, more DMUs are in general classified as efficient, as the excel in one of the many dimensions. To check for this, we eliminated number of codefiles and total LOC from the list of output factors, leaving one code-related measure only. Results indeed indicate some change, but still a large amount of projects is deemed to be efficient:

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.9200	0.9993	1.0000	0.9952	1.0000	1.0000

Of course, the current data set is gravitating towards larger projects of higher success rate (including e.g. GNOME, audacity or squirrelmail), so there might be a smaller variance in efficiency as compared to a more diverse set as later available.

Currently, the efficiency scores are added to the data set. This allows to make additional analyses in the future, used the efficiency score as a variable, e.g. the dependent variable in a model with other project dependent measures as independent variables. This could include process related measures, but also age or license type. For the time being, FLOSSMetrics has shown to be a data source which can easily provide the data necessary for this type of study.

3.7.4 References

- Banker, R.D., Chang, H., and Kemerer, C.F. (1994). Evidence on economies of scale in software development. *Information and Software Technology*, 36(5), 275-282.
- Banker, R.D., Charnes, A., and Cooper, W. (1984). Some Models for Estimating Technical and Scale Inefficiencies in Data Envelopment Analysis. *Management Science*, 30, 1078-1092.
- Banker, R.D. and Kauffman, R.J. (1991). Reuse and Productivity in Integrated Computer-Aided Software Engineering: An Empirical Study. *MIS Quarterly*, 15(3), 375-401.
- Banker, R.D. and Kemerer, C. (1989). Scale Economies in New Software Development. *IEEE Transactions on Software Engineering*, 15(10), 416-429.
- Banker, R.D. and Slaughter, S.A. (1997). A Field Study of Scale Economies in Software Maintenance. *Management Science*, 43(12), 1709-1725.
- Boehm, B.W., Abts, C., Brown, A.W., Chulani, S., Clark, B.K., Horowitz, E., Madachy, R., Reifer, D.J., and Steece, B. (2000). *Software Cost Estimation with COCOMO II*. Upper Saddle River, NJ: Prentice Hall.
- Charnes, A., Cooper, W., and Rhodes, E. (1978a). *A Data Envelopment Analysis Approach to Evaluation of the Program Follow Through Experiments in U.S. Public School Education* (Management Science Research Report No. 432). Carnegie-Mellon University, Pittsburgh, PA.
- Charnes, A., Cooper, W., and Rhodes, E. (1978b). Measuring the Efficiency of Decision Making Units. *European Journal of Operational Research*, 2, 429-444.

- Crowston, K., Annabi, H., and Howison, J. (2003). Defining Open Source Software Project Success. In *Proceedings of ICIS 2003*, Seattle, WA.

- Crowston, K., Annabi, H., Howison, J., and Masango, C. (2004). Towards A Portfolio of FLOSS Project Success Measures. In *Collaboration, Conflict and Control: The 4th Workshop on Open Source Software Engineering (ICSE 2004)*, Edinburgh, Scotland.

- Crowston, K., Howison, J., and Annabi, H. (2006). Information systems success in free and open source software development: theory and measures. *Software Process: Improvement and Practice*, 11(2), 123-148.

- Diaz-Martinez, Z. and Fernandez-Menendez, J. (2008). DEA: Data Envelopment Analysis. R package version 0.1-2.

- Farell, M.J. (1957). The Measurement of Productive Efficiency. *Journal of the Royal Statistical Society*, Series A 120(3), 250-290.

- Kitchenham, B. (2002). The question of scale economies in software - why cannot researchers agree? *Information & Software Technology*, 44(1), 13-24.

- Kitchenham, B. and Mendes, E. (2004). Software Productivity Measurement Using Multiple Size Measures. *IEEE Transactions on Software Engineering*, 30(12), 1023-1035.

- Koch, S. (2004). Profiling an open source project ecology and its programmers. *Electronic Markets*, 14(2), 77-88.

- Koch, S. (2008). Measuring the Efficiency of Free and Open Source Software Projects Using Data Envelopment Analysis. In Sowe, S.K., Stamelos, I., and Samoladas, I. (eds.): *Emerging Free and Open Source Software Practices*, pp. 25-44, Hershey, PA: IGI Publishing.

- Koch, S. and Neumann, C. (2008). Exploring the Effects of Process Characteristics on Product Quality in Open Source Software Development. *Journal of Database Management*, 19(2), 31-57.

- Koch, S. and Schneider, G. (2002). Effort, Cooperation and Coordination in an Open Source Software Project: Gnome. *Information Systems Journal*, 12(1), 27-42.

- Mayrhauser, A., Wohlin, C., and Ohlsson, M. (2000). Assessing and Understanding Efficiency and Success of Software Production. *Empirical Software Engineering*, 5(2), 125-154.

- Michlmayr, M. (2005). Software Process Maturity and the Success of Free Software Projects. In Zielinski, K. and Szmuc, T. (eds.): *Software Engineering: Evolution and Emerging Technologies*, pp. 3-14, Amsterdam, The Netherlands: IOS Press.

- Mockus, A., Fielding, R., and Herbsleb, J. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology,* 11(3), 309-346.

- Myrtveit, I. and Stensrud, E. (1999). Benchmarking COTS Projects Using Data Envelopment Analysis. In *Proceedings of 6th International Software-Metrics-Symposium,* pp. 269-278, Boca-Raton, FL.

- Stewart, K.J. (2004). OSS Project Success: From Internal Dynamics to External Impact. In *Collaboration, Conflict and Control: The 4th Workshop on Open Source Software Engineering (ICSE 2004)*, Edinburgh, Scotland.

- Stewart, K.J., Ammeter, A.P., and Maruping, L.M. (2006). Impacts of Licence Choice and Organisational Sponsorship on User Interest and Development Activity in Open Source Software Projects. *Information Systems Research* 17(2), 126-144.

- Stewart, K. J. and Gosain, S. (2006). The Moderating Role of Development Stage in Affecting Free/Open Source Software Project Performance. *Software Process: Improvement and Practice*, 11(2), 177-191.

3.8 Correlation Size and Complexity Metrics

Note: more information about this study can be found in the paper: "*A study of the properties of libre software"* by Israel Herraiz. This paper is included as annexe to this document.

3.8.1 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 studied a large quantity of source code (about 100,000 files) corresponding to 1293 software projects stored in the aggregated database of FLOSSMetrics [8].

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.

FLOSSMetrics is of course not the only data source that may be used for a study like this. There are large collections of libre software in the form of distributions (for instance, several Linux distributions like Debian, Fedora, Ubuntu, Mandriva, etc.). In this study, I have chosen FLOSSMetrics because it is easy to milk the databases to obtain information regarding a large amount of software projects. In the case of this study, that information is source code metrics. With the quantity of source code measured in FLOSSMetrics (more than 100,00 files, and 13 MSLOC in total, more than 70% 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 study, the first motivation is to find out which independent metrics may be used to characterize size and complexity. It has been also confirmed that all metrics considered follow a statistical distribution (double Pareto), which was already found for a set of source code files obtained from FreeBSD [7, 6].

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. It has been 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.

3.8.2 Methodology

After all the metrics were extracted, some descriptive statistics were obtained. It were 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, it was 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.

The same procedure was repeated, but this time for the logarithm of the metrics. The correlation coefficients showed strong relationships among the logarithm of the metrics.

More over, the statistical distributions appeared to be very close to a normal distribution.

FLOSSMetrics contains a rich set of size and complexity metrics, although complexity metrics are only available for C source code. Because of that, in this study I focus in the case of C, using the whole set provided by FLOSSMetrics, that can be found in the next table:

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), Halstead's length (HLENG), Halstead's volume (HVOLU), Halstead's level (HLEVE), Halstead's mental discriminations (HMD)

3.8.3 Results

All the data needed for this study were obtained directly from the FLOSSMetrics databases.. In particular, from the aggregated database of source code management (SCM) systems, that at the time of writing this (November 2009), it contains 1293 software projects. Nevertheless, the results shown here can be updated easily when new projects are added to that database.

From all the data available in that database, source code metrics only were extracted. The information contained in the database is historical, this is, contains data for different versions of the same files. It was retrieved the values of the metrics for the latest version available of every file in the database.

The 1293 projects contained 108,680 files, being 25,226 files of them written in C. This nearly 25% of the files accounted for more than 70% of the SLOC.

To obtain that sample of files, we used the following query in the aggregated database of FLOSSMetrics:

```
select lang, sloc, loc, ncomment, lcomment lblank, mccabe_min, nfunctions, mccabe_max,
mccabe_sum, mccabe_mean, mccabe_median, halstead_length, halstead_vol, halstead_level,
halstead_md from (select file_id, max(id) as id from metrics group by file_id) as a,
metrics as b where a.file_id=b.file_id and a.id=b.id and lang="ansic" and sloc>0 and
mccabe_min>=0 and lblank>=0;
```

The query obtain the latest available measurements for all the files written in C, avoiding files with null values of the metrics. We focus in C because it is the only language with complexity metrics that is available in the FLOSSMetrics databases. However, the methods shown here can be applied to any other programming language as soon as new metrics become available.

We calculated the correlation coefficients of the different metrics, using the logarithm of their values. For that purpose, we used the GNU R statistical software. The results are shown in the following table (divided in different sections):

	sloc	loc	ncomment	lblank	mccabe_min
sloc	1.00000000	0.97556034	0.75175871	0.613774335	0.022639153
loc	**0.97556034**	1.00000000	0.80580224	0.725514007	0.013465996
ncomment	0.75175871	0.80580224	1.00000000	0.771479742	-0.052651648
lblank	0.61377434	0.72551401	0.77147974	1.000000000	-0.001458482
mccabe_min	0.02263915	0.01346600	-0.05265165	-0.001458482	1.000000000
nfunctions	0.79112196	0.79964684	0.60372007	0.522058082	-0.319312720
mccabe_max	**0.83257778**	0.83555945	0.59614339	0.536879392	0.265308880
mccabe_sum	**0.88711910**	0.88920828	0.64118477	0.568673593	0.103345367
mccabe_mean	0.56893372	0.56391133	0.37552035	0.353028202	0.651276362
mccabe_median	0.32453945	0.31379473	0.17448753	0.192551722	0.826266879
halstead_length	**0.97730755**	0.96040901	0.72911874	0.606648041	0.052160969
halstead_vol	**0.97623238**	0.95648566	0.72524131	0.601616423	0.047937181
halstead_level	**-0.91159066**	-0.89824546	-0.66907697	-0.581958656	-0.152225967
halstead_md	**0.96930902**	0.95161265	0.71701557	0.604949765	0.088035079

	nfunctions	mccabe_max	mccabe_sum	mccabe_mean	mccabe_median
sloc	0.79112196	0.8325778	0.8871191	0.5689337	0.32453945
loc	0.79964684	0.8355595	0.8892083	0.5639113	0.31379473
ncomment	0.60372007	0.5961434	0.6411848	0.3755203	0.17448753
lblank	0.52205808	0.5368794	0.5686736	0.3530282	0.19255172
mccabe_min	-0.31931272	0.2653089	0.1033454	0.6512764	0.82626688
nfunctions	1.00000000	0.6686102	0.8632031	0.2175424	-0.01200383
mccabe_max	0.66861019	1.0000000	0.9371310	0.8421659	0.56014681
mccabe_sum	0.86320314	0.9371310	1.0000000	0.6783935	0.43628282
mccabe_mean	0.21754236	0.8421659	0.6783935	1.0000000	0.85447061
mccabe_median	-0.01200383	0.5601468	0.4362828	0.8544706	1.00000000
halstead_length	0.76972760	0.8217533	0.8734676	0.5740145	0.34294302
halstead_vol	0.76478672	0.8132807	0.8655880	0.5656555	0.33590716
halstead_level	-0.74084123	-0.8656733	-0.9001210	-0.6636602	-0.44741063
halstead_md	0.76937978	0.8474612	0.8939790	0.6125950	0.38384048

	halstead_length	halstead_vol	halstead_level	halstead_md
sloc	0.97730755	0.97623238	-0.9115907	0.96930902
loc	0.96040901	0.95648566	-0.8982455	0.95161265
ncomment	0.72911874	0.72524131	-0.6690770	0.71701557
lblank	0.60664804	0.60161642	-0.5819587	0.60494977
mccabe_min	0.05216097	0.04793718	-0.1522260	0.08803508
nfunctions	0.76972760	0.76478672	-0.7408412	0.76937978
mccabe_max	0.82175329	0.81328075	-0.8656733	0.84746120
mccabe_sum	0.87346757	0.86558800	-0.9001210	0.89397904
mccabe_mean	0.57401452	0.56565548	-0.6636602	0.61259496
mccabe_median	0.34294302	0.33590716	-0.4474106	0.38384048
halstead_length	1.00000000	0.99903403	-0.9325754	0.99183814
halstead_vol	0.99903403	1.00000000	-0.9254851	0.98979385
halstead_level	-0.93257542	-0.92548507	1.0000000	-0.97001812
halstead_md	0.99183814	0.98979385	-0.9700181	1.00000000

In that table, each cell corresponds to the Pearson correlation coefficient between the metrics in the row and the column corresponding to the cell. For the case of the McCabe cyclomatic complexity, we have included some statistics about that metric: minimum, maximum, median, mean and sum. This is because that metric is measured at the function level, and the values shown in the table are at the file level. A file contains several functions.

As the table shows, SLOC is highly correlated with al the Halstead's complexity metrics, and with the maximum and sum of McCabe's cyclomatic complexity. In other words, SLOC is providing as much information as any other of the complexity metrics, which means that SLOC is as good as classical complexity metrics to measure the complexity of files.

SLOC is not as good correlated with the median, mean and minimum values of McCabe's cyclomatic complexity. However, when measuring the complexity of a file, it seems reasonable to assign the maximum cyclomatic complexity in the functions of that file, because that function will be the bottleneck for a full comprehension by a developer of the file.

Regarding the rest of metrics, the low correlation coefficients verify the soundness of this analysis, It is likely that larger files will contain a greater number of comments or blank lines, and the weak correlation coefficients support this claim. However, it is obvious that the number of comments or the number of blanks cannot (or should not) be a good metric for software characterization. The weak correlation coefficients also support this claim.

3.8.4 Conclusions

In this work, we have shown how to find out which product metrics are providing orthogonal information about the properties of software. In our case, we have focused in the case of C, and in the size and complexity dimensions of files written in that programming language. We have used the rich set of metrics that is available in the FLOSSMetrics databases. After querying the

database we have performed a correlation analysis, to show that SLOC (source lines of code) is a complexity metric as good as other classical metrics like McCabe's cyclomatic complexity and Halstead's software science metrics. This has been verified for a large sample of software, with a total of more than 10 MSLOC of source code.

3.8.5 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. “A statistical examination of the evolution and properties of libre software”. *PhD thesis*, Universidad Rey Juan Carlos, 2008. http://purl.org/net/who/iht/phd.

- [7] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles. “Towards a theoretical model for software growth”. In *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR)*, pages 21–28. IEEE Computer Society, 2007.

- [8] I. Herraiz, D. Izquierdo-Cortazar, F. Rivas-Hernandez, J. M. Gonzalez-Barahona, G. Robles, S. Dueñas Dominguez, C. Garcia-Campos, J. F. Gato, and L. Tovar. “FLOSSMetrics: Free / libre / open source software metrics”. In *Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR). IEEE Computer Society*, 2009.

- [9] 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.

- [10] 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.

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

- [12] 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.

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

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

- [15] 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.

- [16] 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.

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

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

3.9 Project Survival

3.9.1 Introduction

Project survivability is useful information for many open source stakeholders. Coordinators of open source projects would like to know the chances for the survivability of the projects they coordinate, and in particular whether their project is doomed to failure. Companies interested in either adopting an open source product in their information technology infrastructure or participating in a project are also interested in knowing how viable a project is. The probability of the continuation or inversely the probability of "short life" of a project is a factor affecting participation in a project. Volunteer programmers might be more interested in entering a project that has high chances to evolve than to fail and be abandoned. Finally, open source survivability would be beneficial for educational purposes. During the last couple of years there are many universities that have incorporated participation in open source projects in their software engineering curriculum. Instructors supervising such courses are highly interested in their students participating in a successful, to be continued, project (Stamelos 2009).

The purpose of this article is to present a novel framework, based on statistical methods, for the analysis of the survivability of open source projects. In order to test our framework we applied the method proposed on a set of open source projects available in the FLOSSMetrics (a European Union funded project) database.

3.9.2 Methodology

3.9.2.1Preparation

In order to perform our analysis, first we collected raw data from the FLOSSMetrics database regarding source code activity. After we performed a second level analysis in order to perform our study. The time slot we chose for our statistical analysis covers 160 months, each month considered to consist of four weeks. The actual dates are from midnight (00:00:01) of January 1st of 1996 to midnight (23:59:59) of March 12th of 2008. The methodology described in the next sub section, studies the survival time which is defined as the time to the occurrence of a predefined, terminal event. This terminal event, for the purposes of our study will be discussed later. This event has a critical meaning depending on the context of the study (death, failure, response to treatment, etc). In our study, we use the term duration time or simply duration to refer to the time from *birth month* to *the death month*.

With the term *birth month* we refer to the very first instance of activity in the project's repository as it is stored in the FLOSSMetrics database. We assume that *birth month* is the first revision in the stored repository URL. Particularly for projects that already existed on the initial date of our

study, we assume that their *birth month* is this date. We discuss this issue specifically in the validity threats section.

Death month refers to the month that the terminal event occurred. In our study, the terminal event can be of two types:

- The project is considered not active and thus it is abandoned.
- The project has suddenly no code activity.

In order to define (a) we have made the assumption that a project is considered to be inactive if it has less than two commits per month. If this happens then the project is considered inactive for this month. If for the next month the project is inactive again, then it is considered *abandoned* and thus terminated. In their study, Evangelopoulos et al. (2009), consider a project dead if there is no activity at all. For (b), we considered a project to be *lost to follow up* if suddenly it has no commits at all. For this case we assume that the stored repository URL in the FLOSSMetrics database is no longer valid, because for some reason the coordinators of the specific project have decided to move either to another location or to another source code management system.

The duration data include apart from the measurement of duration other variables characterising the projects like the type of the project, the language used and the number of committers. The study of duration data is focused on estimation of the probabilities of duration, comparisons between distributions of different project types and identification of variables correlated to duration.

3.9.2.2 Analysis

As mentioned earlier, for our analysis purposes we used statistical methods suitable for duration data, known as survival analysis. The peculiarity of duration data is that the precise duration times of some projects may not be known. This happens when the terminal event has not yet occurred for some projects in the dataset. These cases in a survival dataset are generally called *censored times* but since our data concern software projects we can refer to them as *active* projects. In our study we have four possible cases for projects, according to their duration: The active ones, those being *inactive for 1 month*, those being *inactive for 2 months* and the ones which are *lost to follow up*. Projects characterized as *inactive for 1* month and *inactive for 2 months* are very few in our sample. They exist because they have been characterized as such during the last couple of month in our analysis time slot. Should our analysis go beyond the time mentioned, some of these would be abandoned, some would become active and other projects would enter these two states, too. In our analysis we will consider the most pessimistic view, i.e. only the active projects will be considered censored, while all the others will be considered terminated, even though *lost to follow up* projects can be still active.

Since the starting month of the research is set to 0 and the cut-off month at 159, we give in the following plots (the numbers of the list refer to the figures accordingly) of all durations:

- Fig 3.2, all projects
- Fig 3.3, active projects
- Fig 3.4, inactive for 1 month projects
- Fig 3.5, inactive for 2 months projects
- Fig 3.6, lost-to-follow-up projects
- Fig 3.7, all projects which are not active

Each project's duration is represented by a straight line segment starting from the birth month and terminating at the death month. The line for active projects stops at month 159 since this is the cut off month of the study.

We can see clearly that the active projects are much more than the others and include a large number of projects that were active for the whole period of research (0 to 159 month).

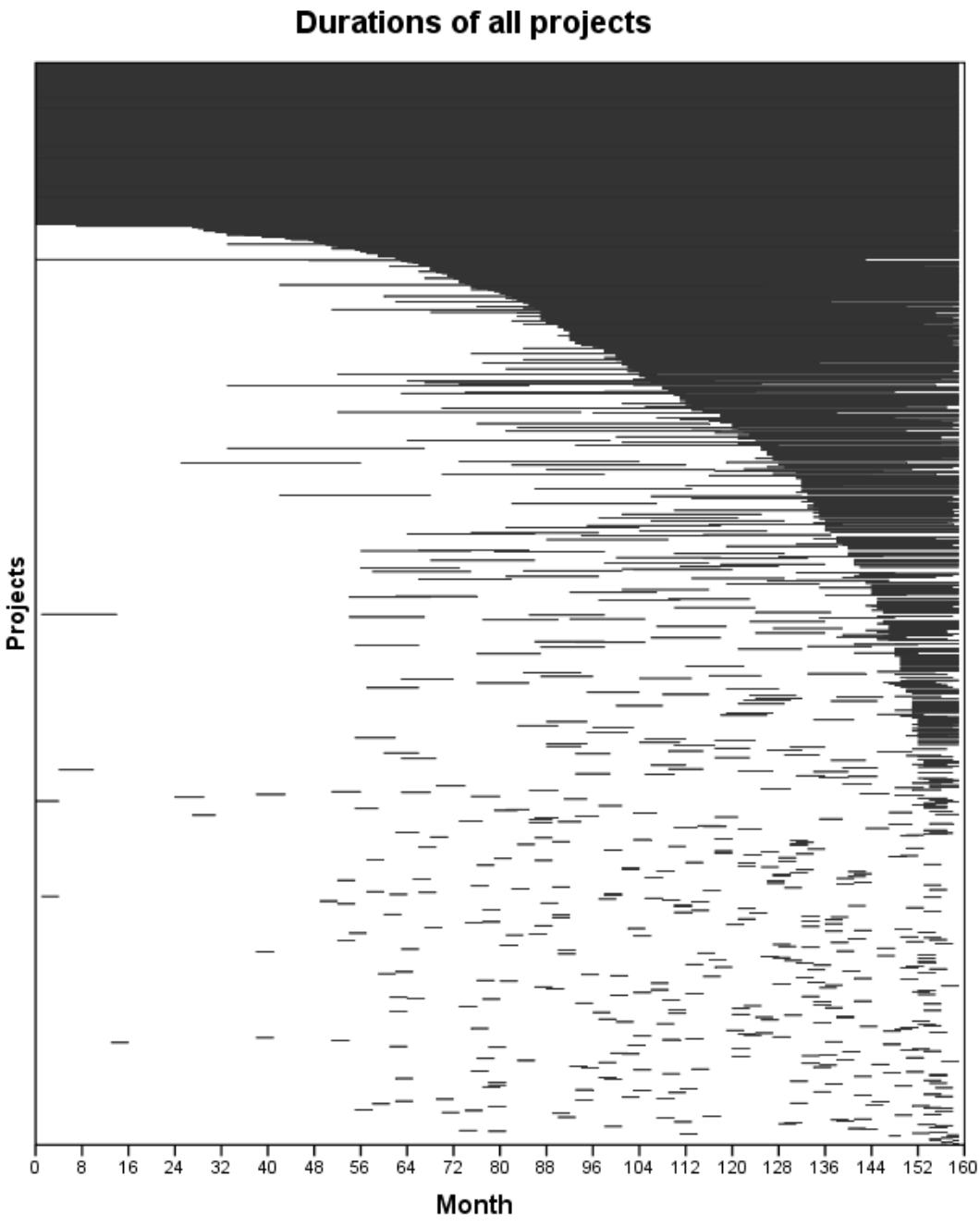


Figure 3.1 Duration of all projects

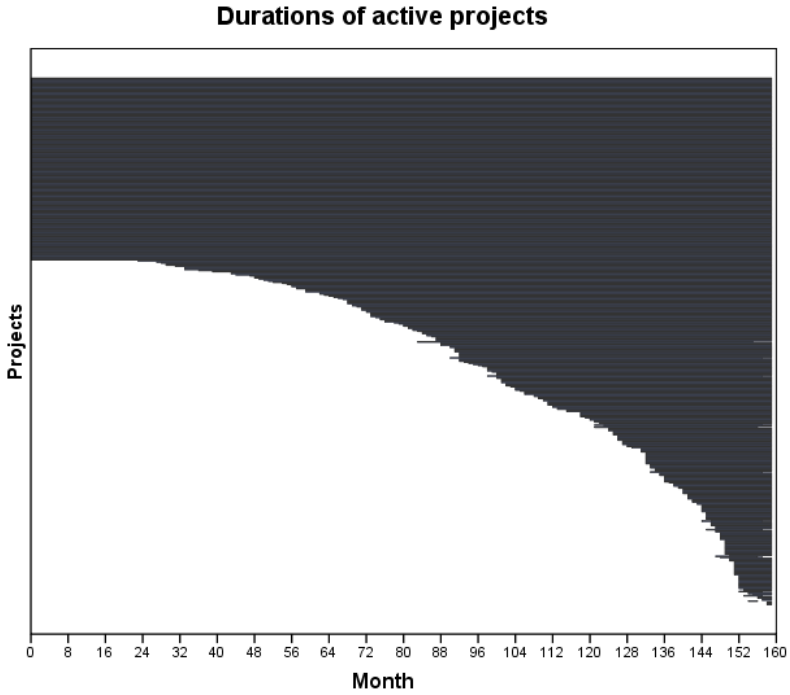


Figure 3.2. Duration of active projects

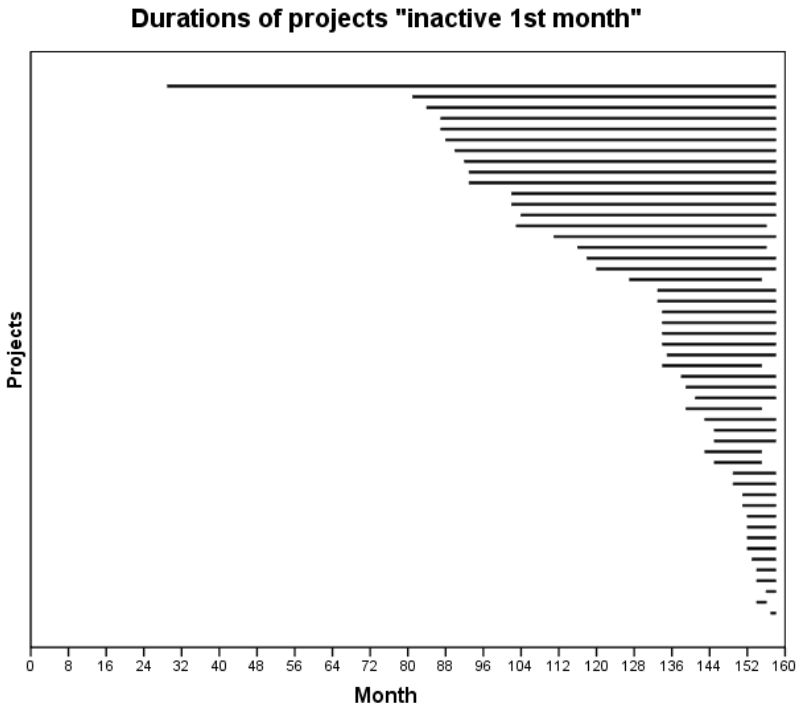


Figure 3.3. Duration of projects characterized as “Inactive 1st month”

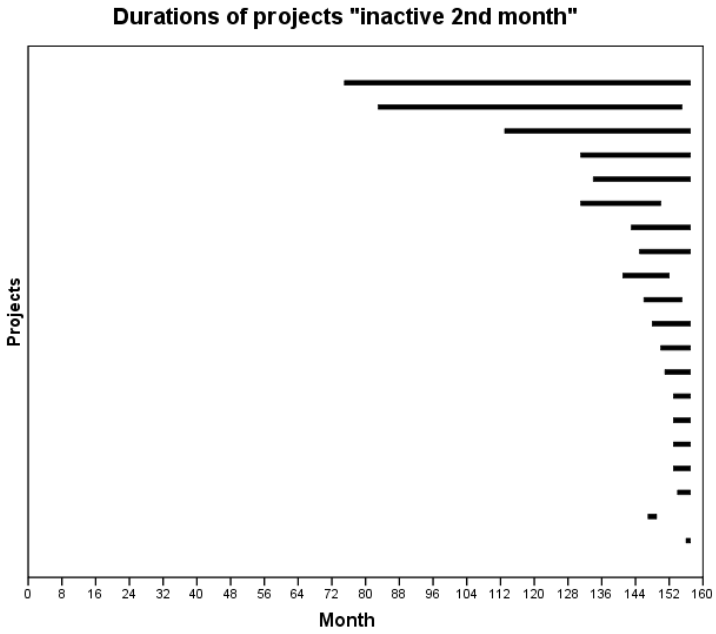


Figure 3.4. Duration of projects characterized as “Inactive 2nd month”

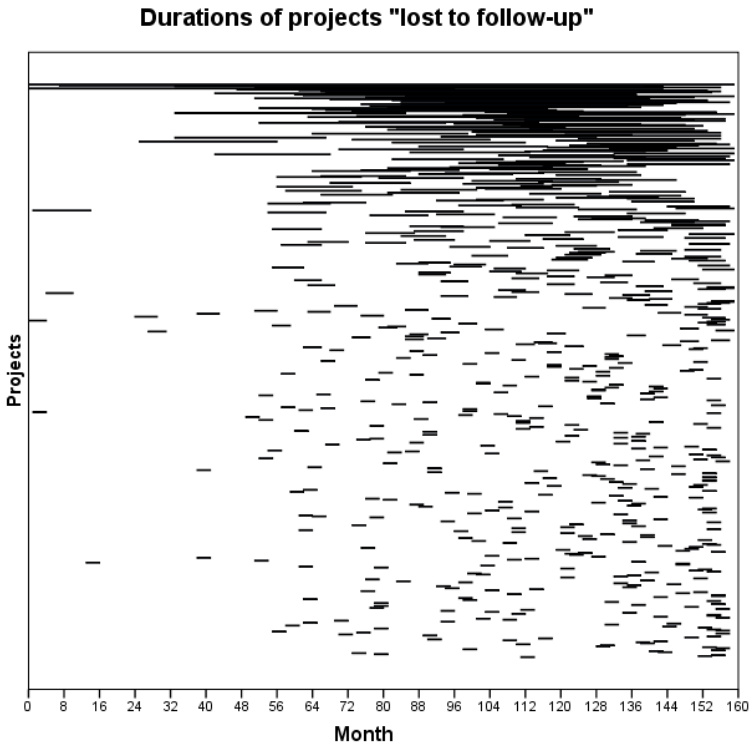


Figure 3.5. Duration of projects characterized as “Lost to follow up”

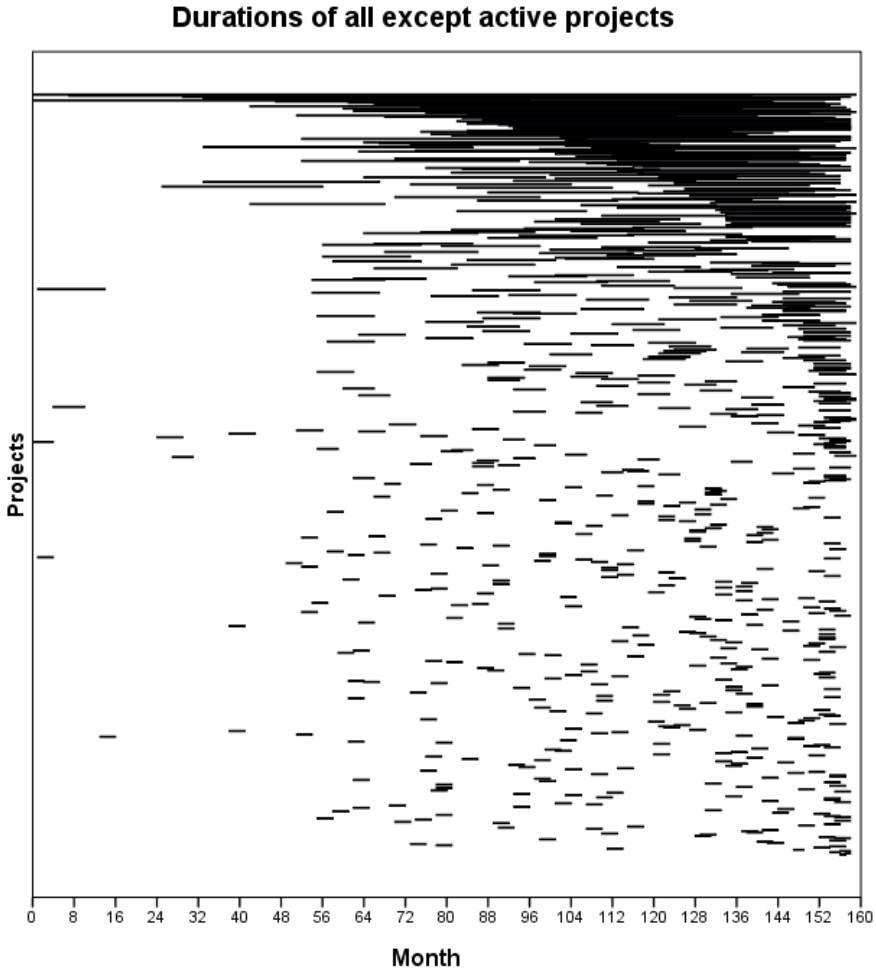


Figure 3.6. Duration of all projects except from active ones.

3.9.3 Results

The distribution of duration can be described by three functions:

- the probability density function (PDF)
- the survival function (SF)
- the hazard function (HF)

These three functions are mathematically equivalent but their interpretation is different. Also, these functions are estimated from the data by approximation methods. In what follows, we denote by T the positive continuous random variable representing the duration time of software projects while by t we denote its values.

The most widely used method for estimating the SF in the presence of censored values is known as *product-limit* (P-L) or *Kaplan - Meier* (K-M) method (Kaplan and Meier, 1958. For the exact methodology please refer to the Annex with this study.

The K-M curve enables us to describe graphically all the durations in the available dataset, even the censored ones.

In the following figures we can see the SF and the Cumulative Hazard Function (CHF) estimated by K-M for all projects. The censored cases are the projects characterized as active. We can see in general long durations even in this pessimistic estimation where all projects that were lost to follow up were considered as terminated. However the probability of durations more than 160 months falls below 40%.

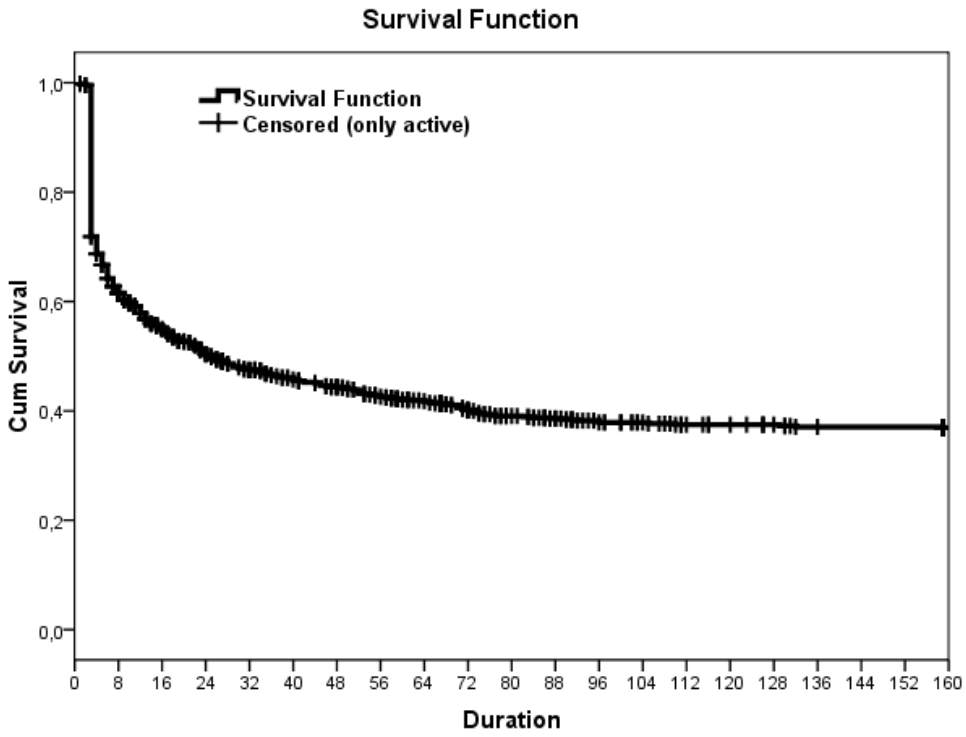


Figure 3.7 Kaplan-Meier estimation of the survival function of all projects. Only the active projects are considered as censored cases

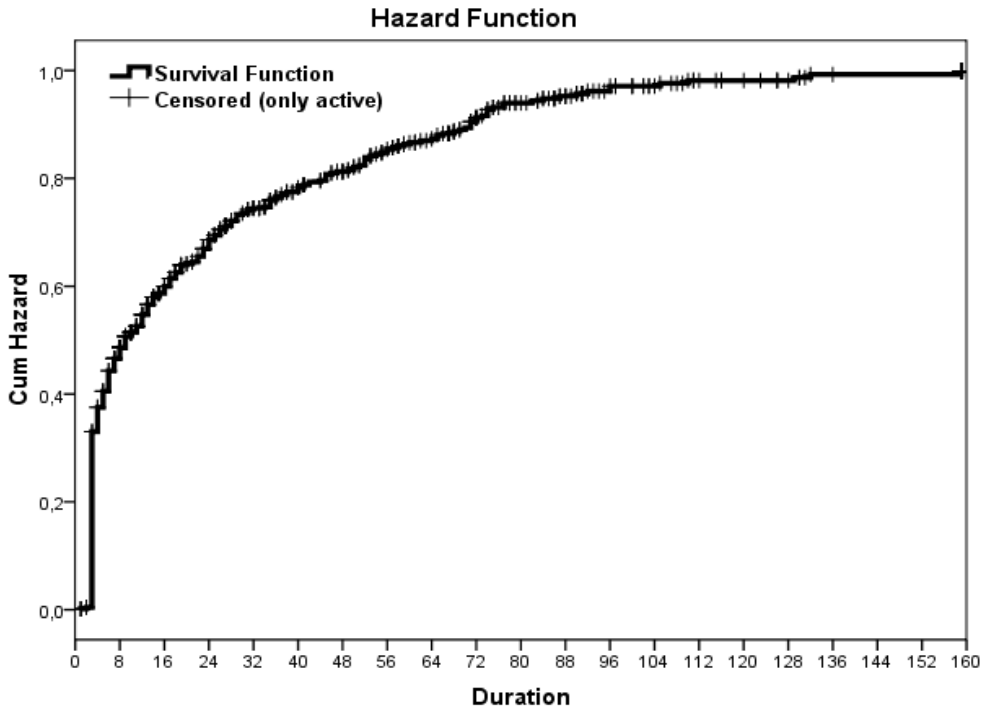


Figure 3.8 Kaplan-Meier estimation of the cumulative hazard function of all projects. Only the active projects are considered as censored cases

In order to see the SF under the most optimistic estimation, we considered all the lost – to – follow up projects as still active. The SF and the CHF are given below. We can see clearly that the probability of survival of a project for more than 159 months does not fall below 80%.

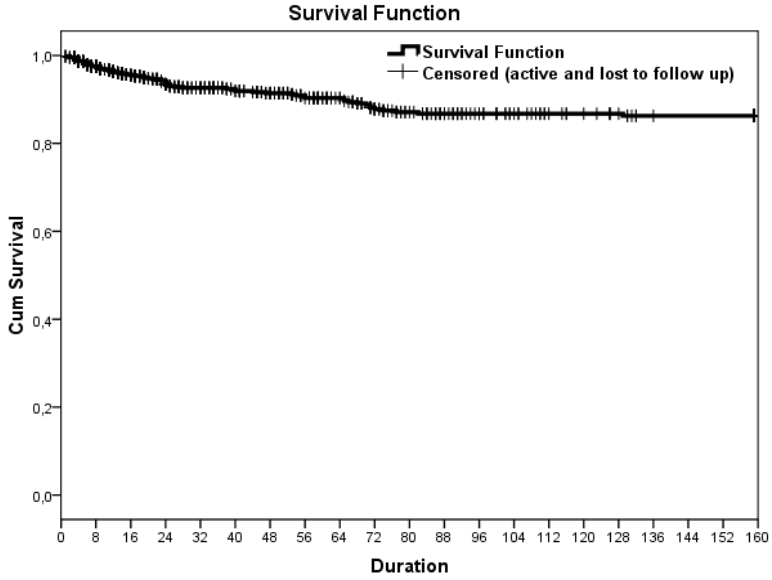


Figure 3.9 Kaplan-Meier estimation of the survival function of all projects. Lost to follow up projects are also considered as censored

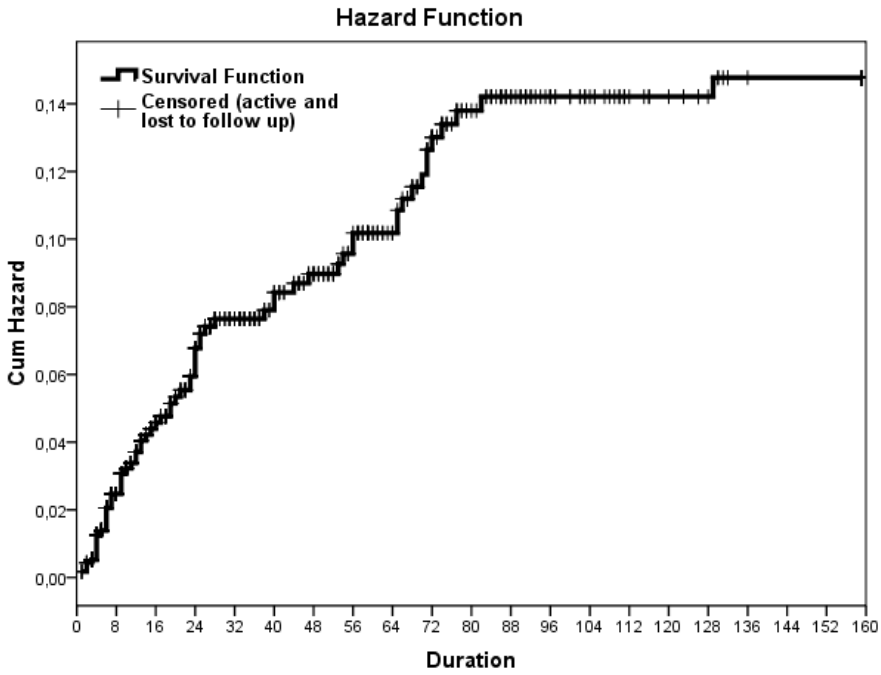


Figure 3.10 Kaplan-Meier estimation of the cumulative hazard function of all projects. Lost to follow up projects are also considered as censored

The SF of all the project types simultaneously is depicted in Figure 11. SF figures of all project types separately are given in the Appendix. Some curves seem to have different behavior (for example see the “software development” which shows higher probabilities of long durations and the “security” and “games and education” with lower durations). This is not strange since software development tools, such as compilers, are often very prolific. However, the tests (such as the log-rank test) did not show significant difference. First we give the distribution of project types (Table 1) and the K-M estimation of their mean and median (Table 2). In the case where censored times are present, the K-M curve is first estimated and the median is found as the value of M satisfying the equation $\hat{D}(M) = 0.5$. The interpretation is quite simple: it is the time in which 50% of the projects will be terminated while the other 50% will be active beyond this time. Median is more reliable measure than mean, due to high skewness. Skewness is inherent in our dataset because of the vividness of the open source world and the multitude of diverse FLOSS projects.

category

		Frequency	Percent
Valid	Communications	88	7,7
	Database	45	3,9
	Desktop	33	2,9
	Education	21	1,8
	Games and Entertainment	92	8,0
	Internet	94	8,2
	Multimedia	122	10,6
	Office	85	7,4
	Science	121	10,5
	Security	17	1,5
	Software Development	215	18,7
	System	214	18,7
	Total	1147	100,0

Table 1. Distribution of project types.

Means and Medians for Survival Time

	Mean	Median
Project Type	Estimate	Estimate
Communications	59,859	16,000
Database	66,516	7,000
Desktop	68,561	26,000
Education	62,664	20,000
Games and Entertainment	57,924	24,000
Internet	63,792	16,000
Multimedia	72,253	38,000
Office	61,896	14,000
Science	79,176	52,000
Security	54,141	9,000
Software Development	81,265	57,000
System	67,349	18,000
Overall	69,520	25,000

Table 2. K-M estimation of means and medians.

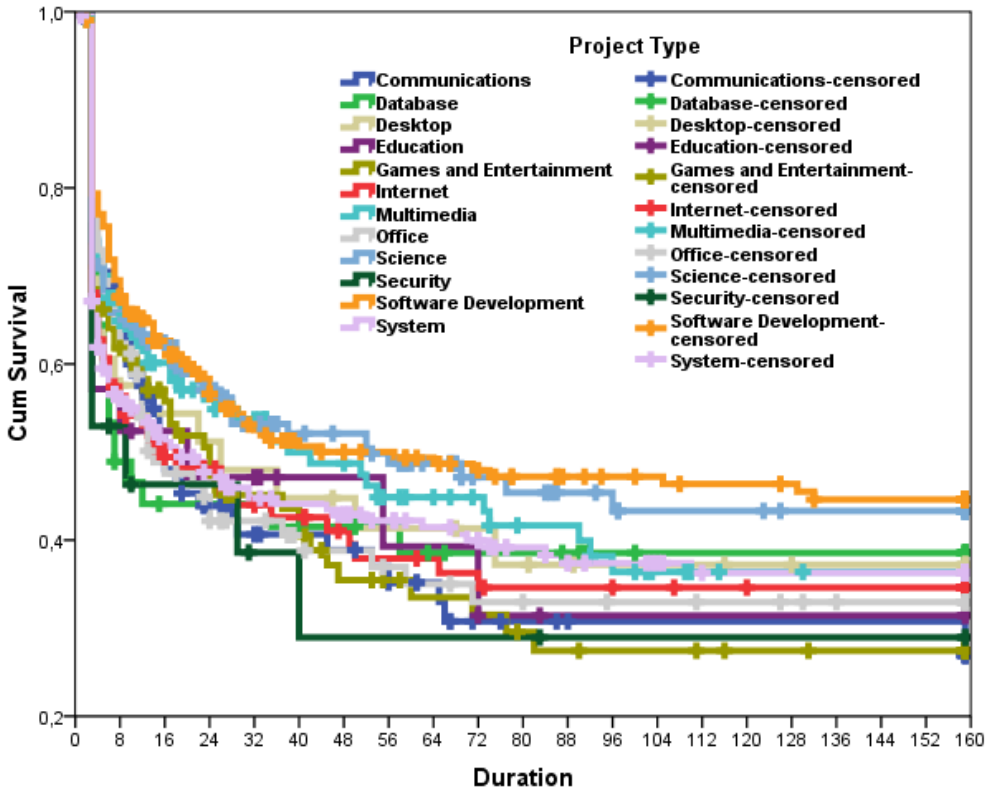


Figure 3.11. Kaplan-Meier estimations of the survival functions, separately for each project type. Only active projects are considered as censored

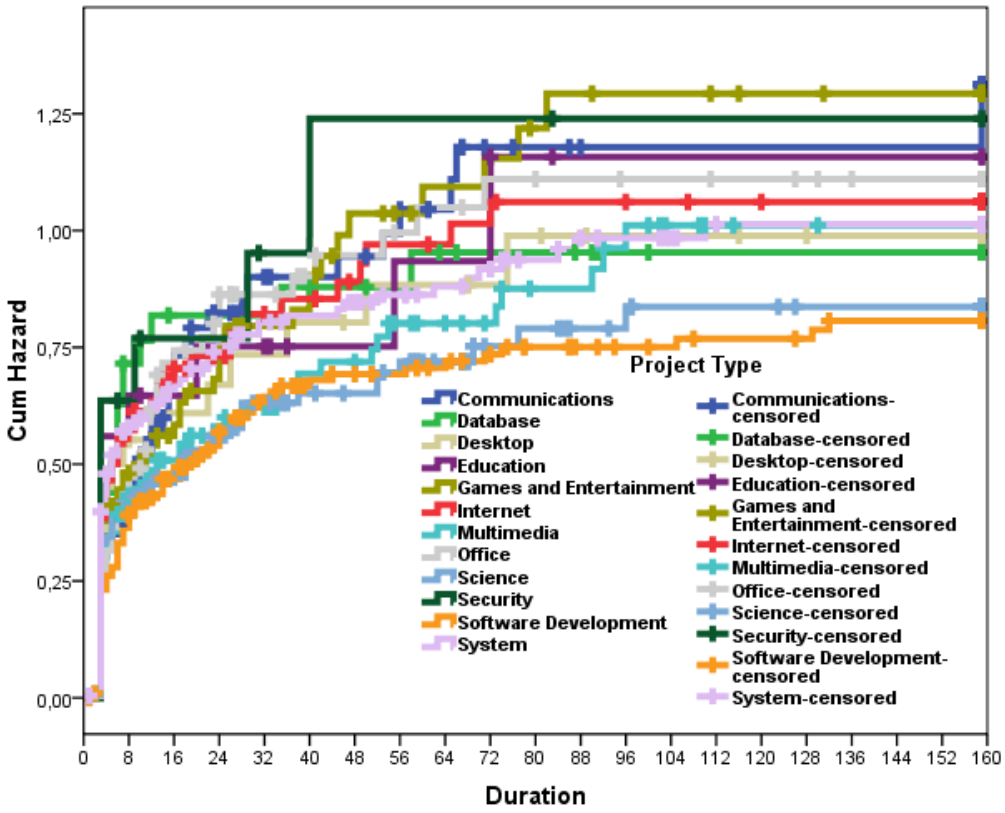


Figure 3.12 Kaplan-Meier estimations of the cumulative hazard functions, separately for each project type. Only active projects are considered as censored

The prediction of the future of a project is called *prognosis*. When data contain a number of project attributes, it is often difficult to identify which ones are most closely related to prognosis.

These variables usually interact in complicated ways and their individual effects are confounded in unpredictable manners. For this reason it is preferable to study the combined effect of all the variables on the duration by modeling this relation.

There are various parametric models for modeling the relationship of duration with other variables but they require the knowledge of an underlying theoretical distribution for the error components of the model. This knowledge is often not available, so the common practice is to use a class of comparative, semi-parametric models for duration data. The most known methodology is the *Cox regression analysis* which is used here.

The HF by definition is an expression of the accumulated knowledge over time, generally known as *aging*. Since HF is applied only to projects that have stayed active up to a particular time, it accounts for the aging that has taken place in the dataset (see Hosmer and Lemeshow, 1999). The point here is that when projects are observed over time, we essentially witness an aging process. It is therefore reasonable to use the HF for modeling the aging process in a dataset.

In our case we have information about two predictors, i.e. the numerical variable *committers* and the categorical variable *project type*. For building the model we used a stepwise Cox regression, which is capable to identify only the most significant predictors for the Cox model. The

algorithm resulted in a model with only one significant variable, namely the committers ($p<0.001$). The categorical variable was not considered significant for the model.

The final model is:

$$h(t;\mathbf{x}) = h_0(t)\exp(-0.172 \times committers)$$

The value $e^{-0.172} = 0.842$ means that the HF is reduced by $100\% - (100\% \times 0.842) = 15.8\%$ each time a unit is added to committers.

It should be emphasized that the Cox regression does not produce point estimations (single values) of the duration of a project. Its prognosis is a whole distribution of duration values with assigned probabilities. So for any value of the predictors we can get interesting graphs of the SF. For example in the following graph we can see the SF curve corresponding to the *average project* in the sense that in Cox regression equation we substitute the variable *committers* by its mean value. In this way we have the curve for a hypothetical project having committers= 10.5.

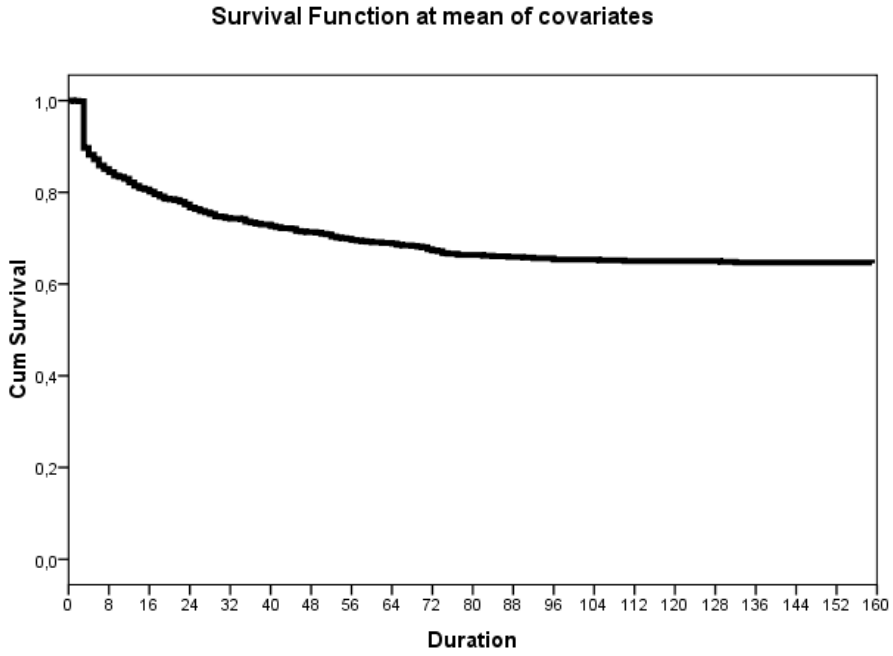


Figure 3.13 Estimation of the survival curve by Cox regression for committers= 10.5

We can see that for about 11 committers the probability of long duration does not fall below 60%. On the other hand, the following figure shows the SF when committers=1. The probability of long duration falls below 20%.

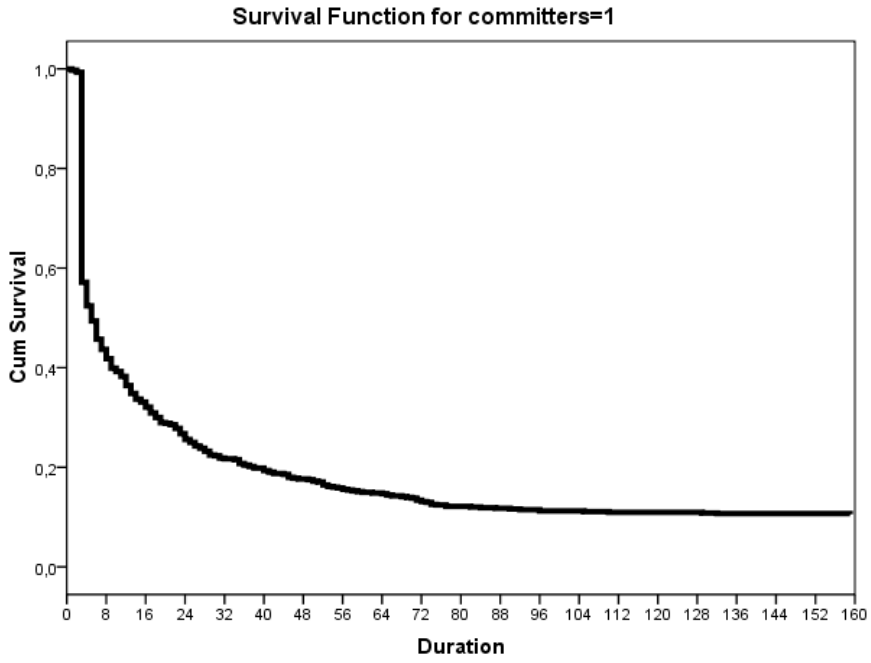


Figure 3.14 Estimation of the survival curve by Cox regression for committers= 1

For committers=20 the curve does not fall below 80%.

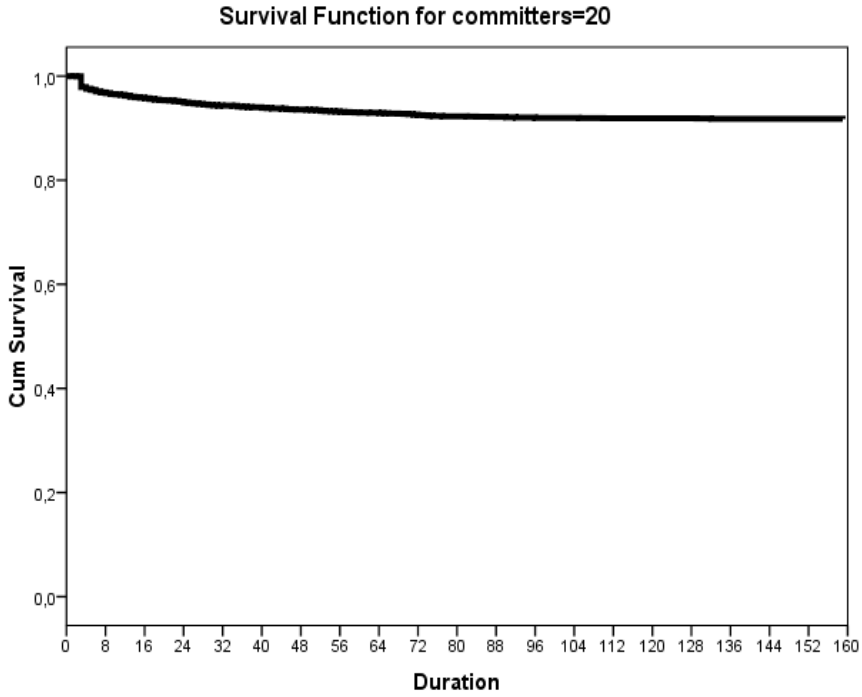


Figure 3.15 Estimation of the survival curve by Cox regression for committers= 20

Looking at the tables and the analysis, a first observation that can be clearly drawn is that projects which have already a prolonged activity and presence in open source landscape are difficult to be abandoned. Figure 3.2 shows that projects that existed more than ten years ago continue to evolve. This could mean that there are open source projects which have proved their value and established a sustainable community.

Regarding the probability of survival of open source projects, it can be argued that with our pessimistic calculations, this is about 40% for more than five years. The optimistic calculation, that with the *lost to follow up* projects included, raises this probability up to 80%. We consider these numbers encouraging.

Although statistical tests do not show any significant difference, Tables 1 and 2 show that there are certain types of software which seem to have higher probability of survival. For example projects in the “Software Development” and “Science” category have more chances to continue to evolve, while projects in “Database” or “Security” less. Maybe this is so because both these categories are crucial and important and users usually choose the most popular projects and avoid trying new ones. In any case this is a finding that deserves better examination.

One other interesting result of our study is the outcome of the Cox regression analysis that showed that for every new programmer (committer) we add in a project, its survivability is increased by 15.8%. This finding does not folow the famous *Brooks Law* that “adding manpower to a late project makes it later”, a fact already observed by other researchers. Scweik et al. (2008) showed that for each developer added to an open source project, the chances for this project to become successful increases 1.24 times. The idea that in open source larger team means successful projects, dubbed as “Linus’ Law” is a hot research issue nowadays. Recently Capiluppi and Adams (2009) conducted a study questioning Brook’s law and found that, regarding communication channels, it is valid for the core developer team of an open source project. Future analysis on open source manpower will give us more insight and would prove a useful tool for open source coordinators.

3.9.4 References

Angelis, L. and Sentas, P. 2005. *Duration Analysis of Software Projects*. Proceedings of the 10th Panhellenic Conference on Informatics, 258--269.

Beecher, K., Capiluppi, A. and Boldyreff, C. *Identifying exogenous drivers and evolutionary stages in FLOSS projects*. The Journal of Systems and Software, 82, (2009) 739 – 750

Capiluppi, A. and Adams, P. *Reassessing Brooks’ Law for the Free Software Community*. Proc. IFIP 5th International Conference on Open Source Software Systems (OSS2009).

Crowston, K., Annabi, H. and Howison, J. *Defining Open Source Software Project Success.* Proceedings of ICIS 2003, Seattle, WA, 14-17 December

Crowston, K., Howison, J., and Annabi, H. *Information systems success in Free and Open Source Software development: Theory and measures*. Software Process--Improvement and Practice, (2006), 11(2), 123–148.

English, R. and Schweik, C. M. *Identifying Success and Abandonment of FLOSS Commons: A Classification of Sourceforge.net Projects*. Upgrade: The European Journal for the Informatics Professional VII.6 (2007).

Evangelopoulos, N., Sidorova, A. Fotopoulos, S. and Chengalur-Smith, I. *Determining Process Death Based on Censored Activity DataI.* Communications in Statistics – Simulation and Computation, 37: 1647 – 1662 (2009)

Hosmer, D. W. and Lemeshow, S. *Regression Modelling of Time to Event Data*. Wiley 1999.

Howison, J. and Crowston, K. *The perils and pitfalls of mining SourceForge*. in 'Proc. of Workshop on Mining Software Repositories at the International Conference on Software Engineering ICSE (2004)

Kaplan, E. L. and Meir, P. *Nonparametric Estimation for Incomplete Observations*. Journal of the American Statistical Association, (1958), 53, 457--481.

Lee, E. T. and Wang, J. W. *Statistical Methods for Survival Data Analysis*. 3rd ed, Wiley (2003).

Koru, Zhang, Liu. *Effect of Coupling on Defect Proneness in Evolutionary Open-Source Software Development*. Proc. IFIP 3rd International Conference on Open Source Software Systems (OSS2007).

Parmar, M. K. B. and Machin, D. *Survival Analysis. A Practical Approach*. Wiley (1995).

Rainer, A. and Gale, S. *Evaluating the Quality and Quantity of Data on Open Source Software Projects*. The First International Conference on Open Source Systems Genova, Italy, July 11 - 15, 2005

Schweik, C. M., English, R., Kitsing, M. and Sandra.H. *Brooks' versus Linus' Law: An Empirical Test of Open Source Projects*. The Proceedings of 9th International Digital Government Research Conference,, (2008).

Sentas, P. and Angelis, L. 2005. *Survival Analysis for the Duration of Software Projects*. Proceedings of the 11th IEEE International Software Metrics Symposium.

Sentas, P. Angelis, L. and Stamelos, I. *A statistical framework for analyzing the duration of software projects*. Empirical Software Engineering (2008), 13:147 – 187

Stamelos, I. *Teaching Software Engineering with Free/Libre Open Source Projects*. International Journal of Open Source Software and Processes (2009), 1: 72-90

Weiss, D. *Measuring Success* of Open Source Projects Using Web Search Engines. In Scotto, M., Succi, G. (Eds). Proceedings of the First International Conference on Open Source Systems (OSS2005). Genova, Italy, pp 93 - 99

3.10 Maintainability Assessment: Committers and Complexity

3.10.1 Introduction

Two of the most acclaimed attributes of open source software are its high availability in projects and the high quality of both these projects and their developers. One of the main aspects of qaulity is maintainability which is the complexity of the software in order to be maintained. So, someone would expect that when a project attracts more committers, then it would be less complex. We want to answer these two questions (a) Is the number of total committers per unit of time affecting the complexity of a project and (b) is the number of distinct committers affecting the complexity of a project?

3.10.2 Methodology

For the purposes of our analysis we used metrics data for 68 gnome projects included in the FLOSSMetrics database. For these projects we wanted to gather data about:

1.Committers.
2.Distinct committers.
3.Complexity metrics.

All these three data were collected in certain timeframes, common in all projects. The timeframe was considered to be one month. So for every month we calculated the aforementioned data. As a complexity metric we used the average McCabe complexity number of all the files in the project for the timeframe measured.

3.10.2.1Data Collection

In order to have a continuous time line and a unified set of the data, we used a methodology presented in the FLOSSMetrics wiki page (http://melquiades.flossmetrics.org/wiki/doku.php?id=addfunctionalities). After constructing the dates table as described above, we gathered our data using the following simple script:

```
   #!/bin/bash

   cat gnome.txt | while read line;
   # $db="fm3_"$line"_cvsanaly2_scm_svn"
   do
   mysql -h host -u user -p -e "SELECT x.myyear,x.mymonth, if(g.key2 is
NULL,0,g.mccabe_mean) mccabe FROM fm3_wildcard_date.tblyearmonth x LEFT JOIN (SELECT
date_format(s.date, '%Y') myyear, date_format(s.date, '%m') mymonth, AVG(m.mccabe_mean)
mccabe_mean, date_format(s.date, '%Y%m') key2 FROM metrics m, scmlog s WHERE s.id =
m.commit_id GROUP BY date_format(s.date,'%Y%m'))g ON x.id=g.key2 WHERE x.id>= (SELECT
date_format(min(s.date), '%Y%m') FROM scmlog s) AND x.id <= (SELECT
```

```
date_format(max(s.date), '%Y%m') FROM scmlog s);" "fm3_"$line"_cvsanaly2_svn_scm" >
$line"_mccabe.tab";

   mysql -h host -u user -p -e "SELECT x.myyear,x.mymonth, if(g.key2 is
NULL,0,g.commits_) commits FROM fm3_wildcard_date.tblyearmonth x LEFT JOIN (SELECT
date_format(s.date, '%Y') myyear, date_format(s.date, '%m') mymonth, COUNT(s.id)
commits_, date_format(s.date, '%Y%m') key2 FROM scmlog s GROUP BY
date_format(s.date,'%Y%m'))g ON x.id=g.key2 WHERE x.id>= (SELECT
date_format(min(s.date), '%Y%m') FROM scmlog s) AND x.id <= (SELECT
date_format(max(s.date), '%Y%m') FROM scmlog s);" "fm3_"$line"_cvsanaly2_svn_scm" >
$line"_commits.tab";

   mysql -h host -u user -p -e "SELECT x.myyear,x.mymonth, if(g.key2 is
NULL,0,g.committers_) committers FROM fm3_wildcard_date.tblyearmonth x LEFT JOIN
(SELECT date_format(s.date, '%Y') myyear, date_format(s.date, '%m') mymonth,
COUNT(s.committer_id) committers_, date_format(s.date, '%Y%m') key2 FROM scmlog s GROUP
BY date_format(s.date,'%Y%m'))g ON x.id=g.key2 WHERE x.id>= (SELECT
date_format(min(s.date), '%Y%m') FROM scmlog s) AND x.id <= (SELECT
date_format(max(s.date), '%Y%m') FROM scmlog s);" "fm3_"$line"_cvsanaly2_svn_scm" >
$line"_committers.tab";

   mysql -h host -u user -p -e "SELECT x.myyear,x.mymonth, if(g.key2 is
NULL,0,g.committers) commiters_d FROM fm3_wildcard_date.tblyearmonth x LEFT JOIN
(SELECT date_format(s.date, '%Y') myyear, date_format(s.date, '%m') mymonth,
COUNT(DISTINCT s.committer_id) committers, date_format(s.date, '%Y%m') key2 FROM scmlog
s GROUP BY date_format(s.date,'%Y%m'))g ON x.id=g.key2 WHERE x.id>= (SELECT
date_format(min(s.date), '%Y%m') FROM scmlog s) AND x.id <= (SELECT
date_format(max(s.date), '%Y%m') FROM scmlog s);" "fm3_"$line"_cvsanaly2_svn_scm" >
$line"_committers_d.tab";

   echo $line;
   done
```

After this step we had a total 272 files of data that we needed to analyze.

3.10.2.2 Statistical Analysis

The next step was to answer our initial questions; we had to perform correlation tests on our data. Since we had no indications about the distribution of our data we chose to calculate Sprearman correlation coefficient (rho). Calculations where done using the R statistical package and using a script below (only shown for one variable):

```
#!/usr/bin/Rscript

# pass the arguments
args <- commandArgs(TRUE)

# output
sink("slocVSmccabe.txt", append=TRUE, type="output", split=TRUE)

# construct file names
sloc_f <- paste(args[c(1)], "_sloc.tab", sep="")
mccabe_f <- paste(args[c(1)], "_mccabe.tab", sep="")
```

```
  # read the data
  sloc_data <- read.table(sloc_f, header=TRUE)
  mccabe_data <- read.table(mccabe_f, header=TRUE)

  # attach variables
  attach(sloc_data)
  attach(mccabe_data)

  sloc <- source_lines_of_code

  # run the correlation test
  cor_s <- cor.test(sloc, mccabe, method="spearman")

  line <- paste(args[c(1)], "p.value", cor_s[[3]], "rho", cor_s[[4]])
  line

  x <- seq(1, length(sloc))

  plot_f <- paste(args[c(1)],".png", sep="")
  png(plot_f)

  par(mar=c(5, 4, 4, 4) + 0.1)
  plot(x, sloc, pch=16, axes=F, xlab="", ylab="", type="p", col="red")
  axis(2, col="red", col.axis="red")
  mtext("sloc", side=2, line=2.5)

  box()
  par(new=T)

  plot(x, mccabe, pch=15, axes=F, xlab="", ylab="", type="p", col="blue")
  axis(4, col="blue", col.axis="blue")
  mtext("Average McCabe", side=4, line=2.5)

  axis(1, pretty(range(x), 10))
  mtext("Number of week", side=1, col="black", line=2.5)

  title_p <- paste(args[c(1)], "Spearman rho = ", round(cor_s[[4]], digits=2),
"p.value", round(cor_s[[3]], digits=2))

  title(title_p)
  dev.off()

  q(status=1)
```

The results are shown in Tables 1, 2, 3 and 4, while in the Annex there are the Scatter plots of all data.

3.10.3 Results

The statistical analysis reveals strong correlation between all four variables (SLOC, Commits, Committers, Distinct Committers) versus McCabe cyclomatic complexity. Almost all projects have a Spearman rho correlation coefficient near one with p values almost zero. This can safely lead to the result that when a project attracts more developers and the code grows, the complexity of that code grows, too.

Table 1. SLOC versus average McCabe of files (per month)

PROJECT NAME	p VALUE	Spearmann rho
alacarte	2.12070700666363e-08	0.779507926421696
bug_buddy	6.56885462334888e-97	0.988068239075195
deskbar_applet	2.40265783762326e-31	0.974094236906326
eel	1.08278896672390e-81	0.989965502705216
ekiga	3.00743986460136e-59	0.969569705785745
eog	2.74218648724067e-62	0.951574999374578
epiphany	3.32069235517086e-55	0.97756351204733
evince	6.02866378296411e-119	0.994805153667672
evolution	9.37272266580097e-92	0.97907096378257
evolution_data_server	8.77558704945224e-78	0.973837052163729
evolution_exchange	7.31963493620597e-37	0.968874055279502
evolution_webcal	2.60120765029268e-109	0.99990260530778
fast_user_switch_applet	1.83469130827491e-45	0.993753082360677
file_roller	4.43328997417191e-57	0.977355031398318
gcalctool	1.03661157696067e-150	0.995982909214464
gconf_editor	3.66425493087432e-81	0.990948095286252
gdm	3.33861850742646e-94	0.986299225490892
gedit	9.8751497750782e-78	0.965373679935012
gnome_applets	2.36402041935672e-93	0.973910513340635
gnome_backgrounds	NA	NA
gnome_control_center	5.19245915228221e-93	0.979991115563831
gnome_desktop	1.46548899601659e-80	0.96168333497033
gnome_doc_utils	1.73895239712647e-53	0.991773978662005
gnome_games	1.98762598211650e-78	0.958841319916949
gnome_icon_theme	NA	NA
gnome_keyring	1.49439432450626e-46	0.975675216578903
gnome_keyring_manager	1.22577152163471e-79	0.998972128314922
gnome_mag	3.9822765772554e-69	0.988549664250907
gnome_media	7.90313097207426e-77	0.956568928875728
gnome_menus	5.24182885825281e-57	0.994693419811144
gnome_netstatus	4.43774058322855e-87	0.998174738379168
gnome_nettool	6.97700500707103e-76	0.998614830039324
gnome_panel	5.14465929757376e-93	0.97472630189633
gnome_power_manager	7.3489934743638e-34	0.979913647456354
gnome_screensaver	3.76336462149877e-40	0.98934622533517
gnome_session	4.25588546573069e-100	0.980030818527233
gnome_speech	1.35635458205836e-84	0.992396426655698
gnome_system_monitor	3.04858252037171e-78	0.988107336860022
gnome_system_tools	3.22589184496478e-74	0.97822823870528
gnome_terminal	5.06100297638095e-65	0.979399017624964
gnome_themes	3.33891976099693e-14	0.711450899596619
gnome_user_docs	NA	NA
gnome_utils	4.64957073698376e-70	0.945434733252494
gnome_volume_manager	4.53666543174225e-60	0.99513045391893
gok	9.08434624272796e-67	0.986918010447422

gtk_engines	4.82453169408438e-126	0.99383254005497
gtkhtml	7.66448790384228e-99	0.983755091292118
gtksourceview	1.58628850468519e-44	0.940770695278136
gucharmap	7.93636102767856e-81	0.99408041350698
libgail_gnome	2.45356470641689e-131	0.999900808411203
libgnomekbd	4.607020764285e-26	0.98167115902965
libgtop	1.33366901450839e-112	0.990052840570533
liboobs	9.93684735276528e-23	0.971072541165999
librsvg	7.41311580338644e-78	0.987879023663818
libsoup	1.54164156352802e-74	0.980443195201853
libwnck	3.24140376171196e-73	0.98533021974732
metacity	1.45032518510184e-72	0.984264549239857
nautilus	1.76142425952340e-90	0.978094306181227
nautilus_cd_burner	1.72005659204675e-65	0.98715321825816
orca	6.18393801532944e-45	0.983718547193442
seahorse	4.93985686430871e-55	0.984875558391831
sound_juicer	8.7517982437389e-45	0.970108241978469
tomboy	1.76963551339635e-37	0.970382378753505
totem	7.15537111954135e-64	0.98459006058908
vino	1.82916661938098e-47	0.986699206833435
vte	5.63574445182345e-39	0.93613808596607
yelp	1.86701262479740e-89	0.977274940881272
zenity	3.41654138357292e-75	0.99600623417105

Table2. Commits versus average McCabe of files (per month)

PROJECT NAME	**p VALUE**	**Spearmann rho**
alacarte	1.86106684741281e-05	0.648922702986384
bug_buddy	2.46048438660434e-38	0.873265678388097
deskbar_applet	1.23129465081102e-31	0.9748451285902
eel	5.4923582249008e-62	0.973440193867936
ekiga	3.26573874495642e-57	0.96632508854755
eog	1.29669192942382e-86	0.981533481247826
epiphany	3.99401290691026e-60	0.983203169033152
evince	4.02809122444668e-56	0.937980687190384
evolution	2.19688759720804e-86	0.974630650794653
evolution_data_server	2.07961917846250e-79	0.975464581323828
evolution_exchange	9.5137237439531e-27	0.929186387082225
evolution_webcal	5.13062176266843e-05	0.49808304140535
fast_user_switch_applet	3.45718628940329e-10	0.760927883939819
file_roller	1.74716448916646e-45	0.956121322907107
gcalctool	2.28695646377099e-89	0.970257139244534
gconf_editor	2.53670406370432e-21	0.79334711590003
gdm	1.61505145844580e-44	0.900607643942404
gedit	1.66104003695521e-72	0.958183622239873
gnome_applets	8.15177420316859e-93	0.97344558091826
gnome_backgrounds	NA	NA
gnome_control_center	2.08919619802930e-68	0.951486919649708

gnome_desktop	2.00213532995102e-51	0.896772245749967
gnome_doc_utils	1.82551229276117e-10	0.711775151267255
gnome_games	1.36688298434497e-89	0.971670307948364
gnome_icon_theme	1.01737917528388e-20	0.810222499844767
gnome_keyring	2.83564282162689e-35	0.94706699419254
gnome_keyring_manager	5.42512998907219e-07	0.594720331077109
gnome_mag	8.69941351102642e-26	0.86092131680094
gnome_media	2.12488958723511e-60	0.92422290533297
gnome_menus	3.38598939015068e-13	0.784021762888684
gnome_netstatus	1.20519980377414e-09	0.642147375183114
gnome_nettool	7.57091368358063e-17	0.837366850776394
gnome_panel	1.21041419570258e-61	0.927377679013336
gnome_power_manager	3.09799190622764e-28	0.964471770903118
gnome_screensaver	1.29845863172212e-38	0.987562773885324
gnome_session	2.9974380739035e-66	0.937917673022986
gnome_speech	2.18337872784772e-80	0.990584312741148
gnome_system_monitor	3.81609217806191e-39	0.916358116223952
gnome_system_tools	2.37177359191058e-44	0.917898545735113
gnome_terminal	7.71683353826152e-45	0.941728808419363
gnome_themes	3.02815555933326e-18	0.778224881883501
gnome_user_docs	NA	NA
gnome_utils	6.21142497557392e-86	0.967990462523158
gnome_volume_manager	1.13771430120499e-09	0.689229804404463
gok	2.52220243486842e-23	0.83844410087693
gtk_engines	1.23784366690767e-60	0.935601193169515
gtkhtml	2.12532033332611e-52	0.912792793411211
gtksourceview	2.30307346855432e-42	0.933691662006279
gucharmap	6.81017366753309e-18	0.773189538499038
libgail_gnome	1.08142834726780e-11	0.696611991776232
libgnomekbd	1.01417880341171e-13	0.898693769854887
libgtop	2.52927811806686e-40	0.862762976559086
liboobs	2.73408993201078e-25	0.979628338324601
librsvg	7.46020483960086e-78	0.987877380967737
libsoup	6.04183220798207e-72	0.978013114050621
libwnck	9.04952101988985e-29	0.858883151971569
metacity	4.15766718678459e-61	0.972255380381504
nautilus	8.94739834395212e-89	0.976714023322977
nautilus_cd_burner	1.26182065076843e-33	0.916763883929138
orca	9.51514279560767e-28	0.934766896644443
seahorse	1.71465880436251e-30	0.921639271060354
sound_juicer	3.43165204450749e-29	0.91432795973981
tomboy	1.93170079420758e-30	0.947647223285434
totem	8.62577531132795e-63	0.983617753756324
vino	2.60753356473373e-21	0.888779896592841
vte	4.15088167775675e-28	0.87903813327813
yelp	1.46503490284222e-59	0.933019271285845

Table3. Total number of committers versus average McCabe of files (per month)

PROJECT NAME	p VALUE	Spearmann rho
alacarte	1.86106684741281e-05	0.648922702986384
bug_buddy	2.46048438660434e-38	0.873265678388097
deskbar_applet	1.23129465081102e-31	0.9748451285902
Eel	5.4923582249008e-62	0.973440193867936
Ekiga	3.26573874495642e-57	0.96632508854755
Eog	1.29669192942382e-86	0.981533481247826
Epiphany	3.99401290691026e-60	0.983203169033152
Evince	4.02809122444668e-56	0.937980687190384
Evolution	2.19688759720804e-86	0.974630650794653
evolution_data_server	2.07961917846250e-79	0.975464581323828
evolution_exchange	9.5137237439531e-27	0.929186387082225
evolution_webcal	5.13062176266843e-05	0.49808304140535
fast_user_switch_applet	3.45718628940329e-10	0.760927883939819
file_roller	1.74716448916646e-45	0.956121322907107
gcalctool	2.28695646377099e-89	0.970257139244534
gconf_editor	2.53670406370432e-21	0.79334711590003
gdm	1.61505145844580e-44	0.900607643942404
gedit	1.66104003695521e-72	0.958183622239873
gnome_applets	8.15177420316859e-93	0.97344558091826
gnome_backgrounds	NA	NA
gnome_control_center	2.08919619802930e-68	0.951486919649708
gnome_desktop	2.00213532995102e-51	0.896772245749967
gnome_doc_utils	1.82551229276117e-10	0.711775151267255
gnome_games	1.36688298434497e-89	0.971670307948364
gnome_icon_theme	1.01737917528388e-20	0.810222499844767
gnome_keyring	2.83564282162689e-35	0.94706699419254
gnome_keyring_manager	5.42512998907219e-07	0.594720331077109
gnome_mag	8.69941351102642e-26	0.86092131680094
gnome_media	2.12488958723511e-60	0.92422290533297
gnome_menus	3.38598939015068e-13	0.784021762888684
gnome_netstatus	1.20519980377414e-09	0.642147375183114
gnome_nettool	7.57091368358063e-17	0.837366850776394
gnome_panel	1.21041419570258e-61	0.927377679013336
gnome_power_manager	3.09799190622764e-28	0.964471770903118
gnome_screensaver	1.29845863172212e-38	0.987562773885324
gnome_session	2.9974380739035e-66	0.937917673022986
gnome_speech	2.18337872784772e-80	0.990584312741148
gnome_system_monitor	3.81609217806191e-39	0.916358116223952
gnome_system_tools	2.37177359191058e-44	0.917898545735113
gnome_terminal	7.71683353826152e-45	0.941728808419363
gnome_themes	3.02815555933326e-18	0.778224881883501
gnome_user_docs	NA	NA
gnome_utils	6.21142497557392e-86	0.967990462523158
gnome_volume_manager	1.13771430120499e-09	0.689229804404463
gok	2.52220243486842e-23	0.83844410087693
gtk_engines	1.23784366690767e-60	0.935601193169515

gtkhtml	2.12532033332611e-52	0.912792793411211
gtksourceview	2.30307346855432e-42	0.933691662006279
gucharmap	6.81017366753309e-18	0.773189538499038
libgail_gnome	1.08142834726780e-11	0.696611991776232
libgnomekbd	1.01417880341171e-13	0.898693769854887
libgtop	2.52927811806686e-40	0.862762976559086
liboobs	2.73408993201078e-25	0.979628338324601
librsvg	7.46020483960086e-78	0.987877380967737
libsoup	6.04183220798207e-72	0.978013114050621
libwnck	9.04952101988985e-29	0.858883151971569
metacity	4.15766718678459e-61	0.972255380381504
nautilus	8.94739834395212e-89	0.976714023322977
nautilus_cd_burner	1.26182065076843e-33	0.916763883929138
Orca	9.51514279560767e-28	0.934766896644443
Seahorse	1.71465880436251e-30	0.921639271060354
sound_juicer	3.43165204450749e-29	0.91432795973981
Tomboy	1.93170079420758e-30	0.947647223285434
Totem	8.62577531132795e-63	0.983617753756324
Vino	2.60753356473373e-21	0.888779896592841
Vte	4.15088167775675e-28	0.87903813327813
Yelp	1.46503490284222e-59	0.933019271285845
Zenity	1.13915726069125e-13	0.739810275563988

Table 4. Number of distinct committers versus average McCabe of files (per month)

PROJECT NAME	**p VALUE**	**Spearmann rho**
alacarte	1.86106684741281e-05	0.648922702986384
bug_buddy	7.45416878124056e-37	0.865088631400278
deskbar_applet	1.19385675941087e-34	0.98145291769206
eel	2.84612940411936e-62	0.973813853815086
ekiga	1.16863074777816e-63	0.975554252967367
eog	3.08578004857923e-81	0.977173620471046
epiphany	1.15248151115350e-58	0.981697582001395
evince	4.78203582491864e-56	0.937794293370634
evolution	2.63698314609926e-89	0.977152527433987
evolution_data_server	7.98345707241e-92	0.984956649479788
evolution_exchange	6.29297775069658e-27	0.930223371840738
evolution_webcal	8.18938462884116e-05	0.486237852492486
fast_user_switch_applet	5.90969846602032e-10	0.754474182645031
file_roller	9.38895428131288e-45	0.95424249280757
gcalctool	5.31163481027669e-90	0.97087129334293
gconf_editor	1.70884001565572e-20	0.783349516624838
gdm	4.46438124077868e-43	0.894530083190112
gedit	2.48800745344877e-69	0.95308662474235
gnome_applets	3.28179869201929e-106	0.982881750548644
gnome_backgrounds	NA	NA
gnome_control_center	1.70256924584355e-68	0.951643165868778

gnome_desktop	2.92981942239882e-49	0.888691318149917
gnome_doc_utils	2.53819292785012e-10	0.707865448087037
gnome_games	4.79705397535041e-85	0.967026217515385
gnome_icon_theme	1.48016284207073e-21	0.819846006518527
gnome_keyring	2.1042956993869e-35	0.947541386209382
gnome_keyring_manager	1.23872799272963e-06	0.579291823626346
gnome_mag	1.03211443961614e-24	0.851564201080511
gnome_media	3.07883640942119e-58	0.918400472984576
gnome_menus	7.29587083252153e-13	0.777236408611899
gnome_netstatus	9.21138780325902e-10	0.645573924477451
gnome_nettool	1.56649461527236e-16	0.832856990007669
gnome_panel	9.00616850299526e-62	0.927695313198704
gnome_power_manager	6.6891910837772e-34	0.979996399008563
gnome_screensaver	5.11513482897274e-41	0.990235676081334
gnome_session	4.05098345264635e-63	0.930941938542058
gnome_speech	9.0738915156796e-82	0.991222723986559
gnome_system_monitor	3.38445657838928e-37	0.907577978672936
gnome_system_tools	9.5933210002556e-44	0.915608122741658
gnome_terminal	1.50431578710218e-44	0.940841795368813
gnome_themes	6.00096967666616e-19	0.787902038618014
gnome_user_docs	NA	NA
gnome_utils	4.89086938462523e-86	0.968101320566565
gnome_volume_manager	5.65321950590409e-09	0.667657664993097
gok	1.73227382718405e-23	0.84005024644647
gtk_engines	1.28909436457916e-59	0.933155544180416
gtkhtml	3.13569306211278e-49	0.901891127206239
gtksourceview	1.21264950311736e-40	0.927431636927872
gucharmap	6.91024707049545e-14	0.705283806243438
libgail_gnome	1.07774108630393e-11	0.696647532297761
libgnomekbd	1.40158517944673e-12	0.880776206798823
libgtop	8.3872160216462e-36	0.836719832038907
liboobs	1.20018830957652e-30	0.99018650304306
librsvg	2.92843748666465e-77	0.987517324372252
libsoup	2.65031017293033e-64	0.968920185101178
libwnck	2.04108141252764e-27	0.848276416229153
metacity	3.95309393115622e-53	0.958715172626202
nautilus	6.31325260017269e-91	0.978441046681842
nautilus_cd_burner	3.67549209324172e-31	0.903440562279823
orca	6.14874111205874e-25	0.917786771461948
seahorse	1.89290836037098e-29	0.915834295859635
sound_juicer	8.43015091378289e-28	0.905722943750329
tomboy	1.02479968840533e-31	0.952805270377607
totem	7.18133389136843e-72	0.990192649949996
vino	6.01416566152475e-20	0.875236853307617
vte	1.38202994443926e-26	0.867466278840232
yelp	2.56699729018246e-59	0.932418475097314
zenity	1.10705342032840e-13	0.740057242144545

3.10.4 References

- I. Stamelos, L.Angelis, A. Oikonomou, G.Bleris. “Code Quality Analysis in Open-Source Software Development”, *Information System Journal, 2nd Special Issue on Open-Source*, Blackwell Science, Vol 12 (1), pp 43-60, 2002
- Samoladas, I., Stamelos, I., Angelis, L., Oikonomou, A.. Open Source Software Development Should Strive for Even Better Code Maintainability. Communications of the ACM, October 2004

3.11 Development vs Communications

Note: this study is based on the paper "*Are FLOSS developers committing to CVS/SVN as much as they are talking in mailing lists? Challenges for Integrating data from Multiple Repositories"* that is included as annexe to this document.

3.11.1 Introduction

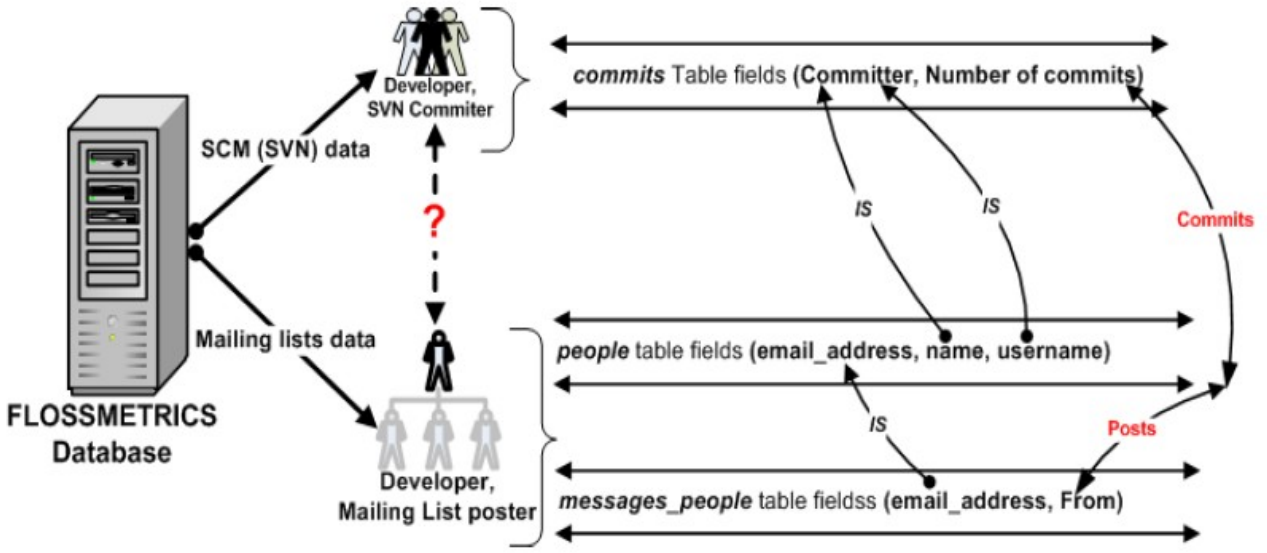
The aim of this study is to decide the degree to which developers participate in mailing lists too. Findings of this study would provide further evidence about the way software knowledge circulates within FLOSS projects. In an early version of this study we used SVN and mailing lists data from 14 projects from the FLOSSMetrics database to accept or reject the hypothesis, "FLOSS developers contribute more to code repository than mailing lists". In that study, we made a number of findings and discovered novel ways to identify and compare developers contribution to both svn and mailing lists. The conclusion of our research supported our hypothesis. The motivation for conducting part 2 of this study is to find out if our findings first study will be better generalized using a large dataset from at least 20 GNOME projects from the FLOSSMetrics database.

FLOSS developers are not bound to a single project. Even where they are contracted to work in corporate, or foundation (Apache) projects, they still have the freedom to participate in other projects or communities of interest. They code, take part in discussions in various mailing lists and forums. Along the way they leave a trail of experience, wealth of knowledge and skills associated with their art. This may be in the form of large and small bits of code, coding ethics and guidelines, externalized explicit knowledge in mailing lists, etc. A number of FLOSS researchers have studied these repositories. However, not all the developers who commit or make changes to a project's source repository also participate in developer mailing lists. We also know the contribution FLOSS make to discussions in mailing lists; they interact with other software developers and users, the keep abreast with project activities and monitor the what goes on in there projects. Little or no research has attempted to correlate developers commits activities with their corresponding mailing lists activities, either within the same project or across projects. The cited research obstacle has to deal with the difficulty in making sure that the developer referenced in CVS/SVN is the same person referenced in the mailing list. Our pilot study found some unique tuples in the FLOSSMetrics SVN and mailing lists database tables which enabled use to match the unique occurrence of developers in both SVN and mailing lists

Overall, this study investigates the concurrence or simultaneous occurrence of FLOSS developers in both SVN and developer mailing lists. That is, we find out if FLOSS developers are coding through commits in SVN as much as they are "talking" in developer mailing list.

3.11.2 Methodology

The methodology employed investigates FLOSS developers activities in both source code and mailing lists repositories aims to overcome challenges associated with identifying the simultaneous occurrence of developers in SVN and mailing lists. This has to do with difficulty in making sure that the developer making SVN commits in a project is the same individual posting to the developer mailing list(s) of the same project. Schematically, the methodology is as shown below:



Some filtering and other techniques were applied on the data:

- Source Code Management Systems
  - Identified and categorized commits as deletions, additions and modifications

- Mailing Lists
  - Removing messages with “Subscribe” and “Unsubscribe” header messages
  - Removing postings with empty header messages and messages with headers but no contents
  - Unmasking aliases in which we identified individuals using different email addresses or names

3.11.3 Results

The sources used were:

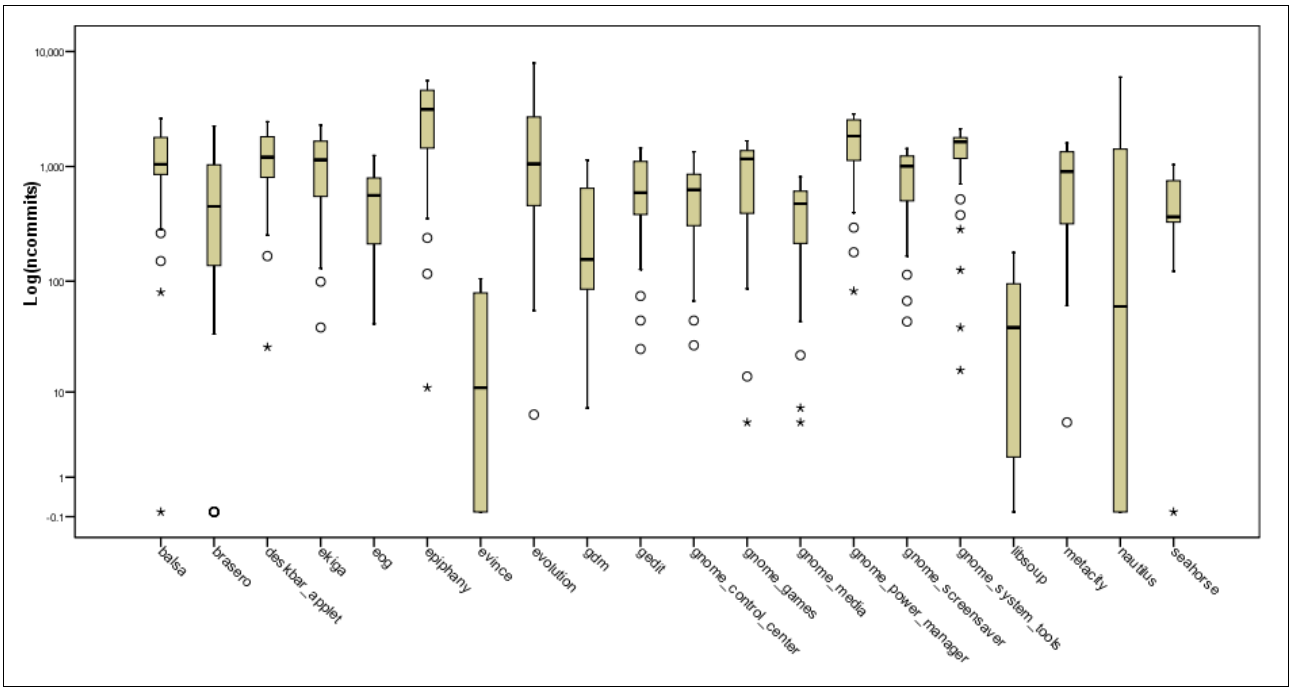
- SVN data from the *second level* (aggregated) of the FLOSSMetrics database (*fm3_aggregatedb_scm)*. This was used to obtain metadata on developers commit_ID and total_number_of_commits in SVN of 20 projects

- Mailing lists data from the *second level* (aggregated) of the FLOSSMetrics database (*fm3_aggregatedb_mls).* This was used to obtain metadata on developers as posters and total_number_of_posts in mailing lists of the *same* 20 projects.

Descriptive statistics of the data from these two repositories and the distribution of commits and posts per project are illustrated below.

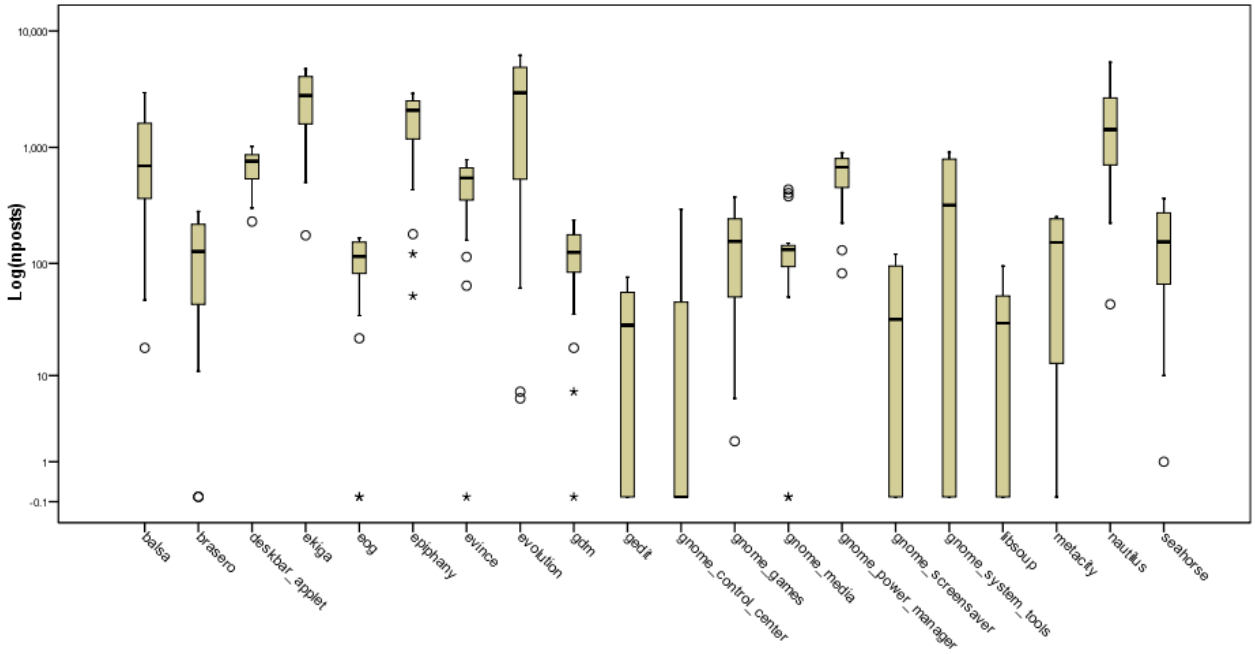
Descriptive statistics of commits

	N	Mean	Median	Std. Dev.	Max.	Sum
balsa	139	4504.97	4418.00	2764,49	8116	626191
brasero	37	681.22	450.00	685.412	2248	25205
deskbar_applet	49	1339.22	1214.00	861.634	2642	65622
ekiga	98	3185.50	2747.00	2338.235	7863	312179
eog	121	2160.56	2071.00	1554.436	5106	261428
epiphany	82	5001.89	5537.50	2874.648	8959	410155
evince	126	984.06	199.50	1208.362	3613	123992
evolution	141	19354.65	21699.00	13396.851	37529	2729005
gdm	126	2625.21	2403.00	2164.281	6808	330776
gedit	138	3026.42	2823.50	2160.869	6991	417646
gnome_control_center	140	4139.29	4178.50	3049.256	9413	579500
gnome_games	143	3546.73	2994.00	2493.177	9068	507183
gnome_media	143	1988.93	1965.00	1377.278	4328	284417
gnome_power_manager	51	1829.67	2115.00	1102.375	3399	93313
gnome_screensaver	54	940.72	1103.00	553.849	1660	50799
gnome_system_tools	110	2604.42	2760.50	1285.477	4342	286486
libsoup	41	62.63	42.00	62.273	205	2568
metacity	101	2024.81	1998.00	1291.698	4239	204506
nautilus	140	8054.23	9539.50	5160.324	15188	1127592
seahorse	81	1171.73	1004.00	870.224	2977	94910



Descriptive statistics of posts

	N	Mean	Median	Std. Dev.	Max.	Sum
balsa	133	9498.50	11734.00	5067,75	14622	1263300
brasero	26	124.69	127.00	89.351	281	3242
deskbar_applet	48	882.46	997.00	265.379	1152	42358
ekiga	38	3798.03	3920.00	2029.302	6975	144325
eog	103	242.29	217.00	175.178	584	24956
epiphany	82	3806.48	4287.50	1633.908	5846	312131
evince	57	893.32	872.00	445.094	1679	50919
evolution	117	19092.88	20515.00	12193.500	36636	2233867
gdm	115	1058.97	848.00	852.725	2721	121781
gedit	114	798.86	450.50	800.887	2488	91070
gnome_control_center	118	444.75	2.50	573.130	1589	52480
gnome_games	80	825.90	777.00	585.760	1726	66072
gnome_media	108	972.77	1145.00	613.671	1756	105059
gnome_power_manager	45	801.69	836.00	294.702	1180	36076
gnome_screensaver	48	40.48	.00	54.903	153	1943
gnome_system_tools	117	988.20	1159.00	719.350	1796	115619
libsoup	41	62.63	42.00	62.273	205	2568
metacity	49	142.35	158.00	118.613	290	6975
nautilus	114	11200.86	12766.50	6154.400	18882	1276898
seahorse	28	186.04	176.00	128.235	386	5209



We investigate our first hypothesis (F/OSS developers make more commits to a project's code repository than they are posting messages to mailing lists) by looking at the descriptive statistics and distribution of commits and posts in the various projects. The table below shows, for each project, the number (Ndev) of developers who made both SVN commits and posted email messages to the project's mailing lists. For the 20 projects, 502 developers made more commits (mean = 152.1; Std. deviation = 431.171) than posts (mean = 43.19; Std. deviation = 164.353).

Developers contribution to both SVN and Mailing Lists

Project	N_{dev}	mean	media	Std. Dev.	Max	Sum	mean	median	Std. Dev.	Max	Sum
balsa	40	37.23	5.5	133.76	851	1489	112.53	25	206.33	751	4501
brasero	6	19.33	2	30.936	77	116	69.17	8.5	98.3	196	415
deskbar_applet	8	20.13	6	35.64	106	161	120.25	5.5	289.5	834	962
ekiga	4	438.25	121.50	722.49	1509	1753	2170.25	2417	1876.01	3757	8681
eog	16	18.81	4.5	25.95	78	301	129.38	37.5	196.16	581	2070
epiphany	40	55.73	7	116.69	470	2229	146.82	16.5	536.03	3352	5873
evince	18	27.17	2	58.61	238	489	100.89	10.5	180.31	535	1816
evolution	92	56.47	4.5	172.68	1481	5195	283.29	46	622.01	4061	26063
gdm	21	26.38	2	56.63	227	554	112.9	17	242.76	939	2371
gedit	19	20.84	2	69.34	306	396	103	4	267.13	1153	1957
gnome_control_center	35	21	4.00	51.753	296	735	69.54	19	125.13	527	2434
gnome_games	14	43.57	7	83.15	304	610	178.93	13.5	341.49	1164	2505
gnome_media	23	21.87	5	36.67	130	503	39.22	6	84.03	345	902
gnome_power_manager	7	3.43	4	1.51	6	24	8	2	11.4	32	56
gnome_screensaver	4	15.75	10	15.84	39	63	211.5	3.5	417.67	838	846
gnome_system_tools	22	17.14	3	33.42	154	377	92.59	24	228.27	1043	2037
ibsoup	3	28.33	8	39.63	74	85	219.67	8	370.09	647	659
metacity	10	14.8	6	23.85	77	148	184.2	7	270.12	600	1842
nautilus	136	49.95	8.5	225.14	2475	6793	86.63	13	220.1	1712	11782
seahorse	2	73.5	73.5	101.12	145	147	198	198	275.77	393	396

The domination of SVN commits, with larger means of commit per developer, over mailing lists posts is evident in all the projects. This means that, indeed, developers are making more commits than they are posting messages to mailing lists. Furthermore, nonparametric correlations were used to study the relationship between commits and posts. There is a low correlation between commits and posts, with Spearman's coefficient () = 0.411 (p =1.000). Wilcoxon signed ranks for the Two-Related-Samples Tests procedure was used to compare the distributions of two5 variables. The results of the test shows that for the 502 developers in the 20 projects; for 140 = ncommits < nposts, for 327 developer ncommits > nposts, and 35 developers had a balanced activity with ncommits = nposts.

3.11.4 Selected References

- Sulayman K. Sowe, Ioannis Samoladas, Ioannis Stamelos, Lefteris Angelis (2008). "Are FLOSS developers committing to CVS/SVN as much as they are talking in mailing lists? Challenges for Integrating data from Multiple Repositories". *3rd International Workshop on Public Data about Software Development (WoPDaSD)*. September 7th - 10th 2008, Milan, Italy.
- J. Anvik and G. C. Murphy. Determining implementation expertise from bug reports. In MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories, pages 1–8. IEEE Computer Society, 2007.
- K. Chen, S. R. Schach, L. Yu, J. Offutt, and G. Z. Heller. Open-source change logs. Empirical Softw. Engg., 9(3):197–210, 2004.
- K. Crowston, A. Hala, and J. Howison. Defining open source software project success. In Proc. of International Conference on Information Systems, ICIS 2003, 2003.
- J.-M. Dalle and M. den Besten. Different bug fixing regimes? a preliminary case for superbugs. In J. Feller, B. Fitzgerald, W. Scacchi, and A. Sillitti, editors, Open Source Development, Adoption and Innovation, volume 234 of IFIP International Federation for Information Processing, pages 247–252.Springer, September 7-10 2007.
- T. T. Dinh-Trong and J. M. Bieman. The freebsd project: A replication case study of open source development. IEEE Transactions on Software Engineering, 31(6):481–494, 2005.
- K. Fogel. Producing Open Source Software: How to Run a Successful Free Software Project. CreativeCommons, 2005.
- D. M. German. Using software trails to reconstruct the evolution of software. Journal of software maintenance and evolution: Research and Practice, 16(6):367–384.
- G. Gousios, E. Kalliamvakou, and D. Spinellis. Measuring developer contribution from software repository data. In MSR '08: Proceedings of the 2008 international workshop on Mining software repositories, pages 129–132. ACM, 2008.
- Q. Jones, M. Moldovan, D. Raban, and B. Butler. Empirical evidence of information overload constraining chat channel community interactions. In CSCW '08: Proceedings of the ACM 2008 conference on Computer supported cooperative work, pages 323–332, New York, NY, USA, 2008. ACM.

- S. K. Sowe, G. S. Ioannis, and M. S. Ioannis, editors. *Emerging Free and Open Source Software Practices.* IGI Global, 2007.

3.12 Contributor Activity

3.12.1 Introduction

In the dynamic FLOSS development process simulation model introduced by Antoniades *et al* [1,2] and applied to real FLOSS projects, several of the input parameter values that were used were merely based on assumptions, due to lack of real data from FLOSS cased studies that had been published by that time. The general aim of this study is to exploit the vast amounts of data on FLOSS projects source code repositories in order to ameliorate this problem for as many of the simulation model input parameters as possible. This will have the effect of making more realistic applications of the simulation model on real life projects, and thus improve its prediction accuracy for the time evolution of key project factors, more specifically LOC added/LOC deleted.

One of the parameters used as input in the dynamic FLOSS development process simulation model by Antoniades *et al* [1,2] is the time dependence of each individual active programmer/contributor productivity, where productivity can, for example, be measured as the number of commits to the project repository or Lines Of Code (LOC) added/deleted. Due to lack of real data, in the above cited works, it was assumed that the productivity of any active contributor, who is not a member of the project core (administrator) group, first increased up to a maximum and then gradually decreased down to zero. In certain cases, the total time period between the first contribution of a contributor, and the point in time when a contributor leaves the project (i.e. becomes inactive for a very long period of time) could be determined *on the average* for all contributors and all types of contributions (functional addition to code, bug fix, bug report) where published case studies for a particular FLOSS project provided data about the average total length of involvement of contributors with the project, (time difference between their first and their last reported commits). However, this was only a very crude estimate of how individual productivity varies in time, and the fact that it should first increase and then decrease (as assumed in was in large an assumption. Also, in past case studies there has been no reported evidence on how the productivity of individuals varies in between their historical first and last commits. . In Figure 1 we show a characteristic curve of the productivity (in arbitrary units) vs time since first commit, as it was assumed in [1,2] for the purposes of applying the simulation model.

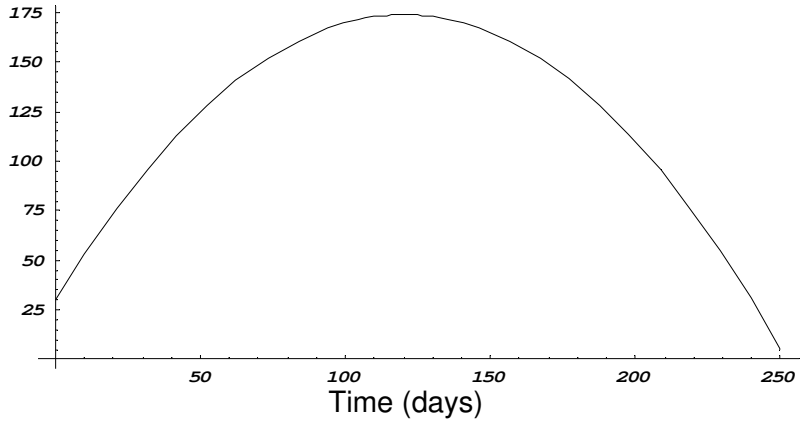


Figure 1: Contributor productivity dependence on time since first commit as was assumed in past applications of the simulation model. Productivity is expressed in arbitrary units in this plot, but is essentially proportional to the number of commits to the project repositories a contributor makes within a particular time period.

By looking at realistic project data provided by FLOSSMETRICS source code repository analysis of a large number of projects, the objective of the present study is to discover the average quantitative dependence of individual contributor productivity on time.

The expected results will be to provide the simulation model with realistic data for the time dependence of contributor behavior and thus strengthen the simulation model's applicability.

3.12.2 Methodology

The approach for obtaining the data was to apply proper composite queries to the SVN FLOSS project repository logs which contain the necessary information (eg. Commiter, time stamp, file(s) affected etc) in cross-reference with the code files themselves, where needed.

We used eight (8) FLOSS projects chosen by the criterion of size, i.e. we chose projects from FLOSSMETRICS database for which SVN log files extended for a long period of time and thus had a large number of commits. In this way, better statistics is possible. These projects were:
1) Python, 2) Evolution, 3) BCLcad, 4) gtk (accumulation of all modules), 5) gtk-gnutella, 6) UfoAI, 7) Plone and 8) Gimp.

After raw data was obtained from FLOSS code repositories, distributions of the above mentioned variables were constructed by using a custom made C console application. As SVN repository log files unfortunately do not report directly the LOC *change* in a file or files per commit entry, this change had to be calculated by special manipulation of the log file data which is performed by the C application.

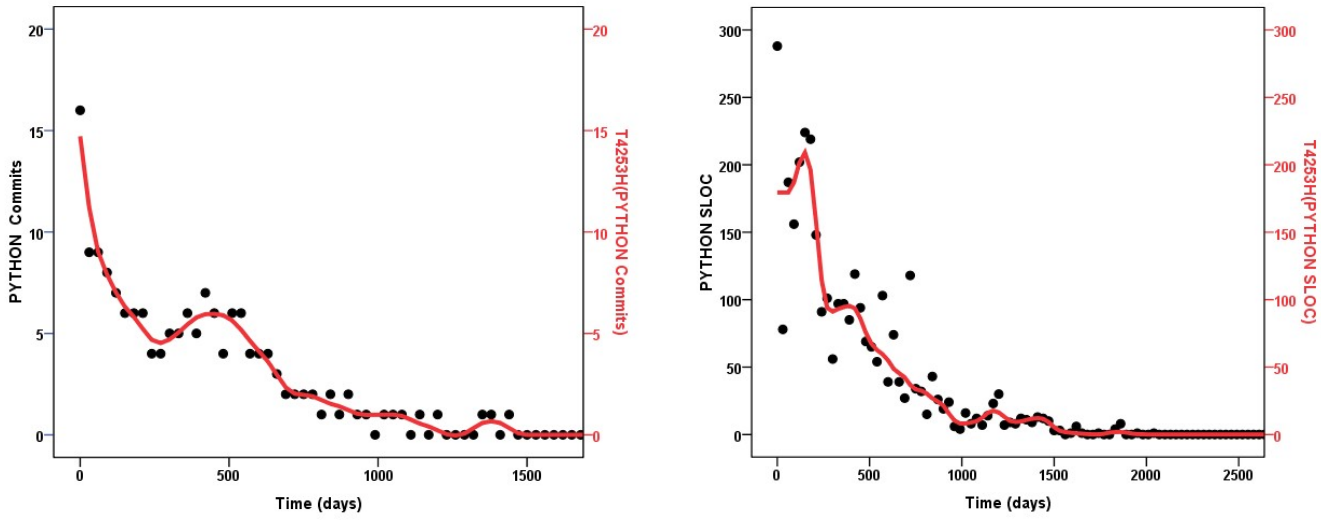
Further statistical and numerical analysis of the distributions and raw data was performed via the SPSS package.

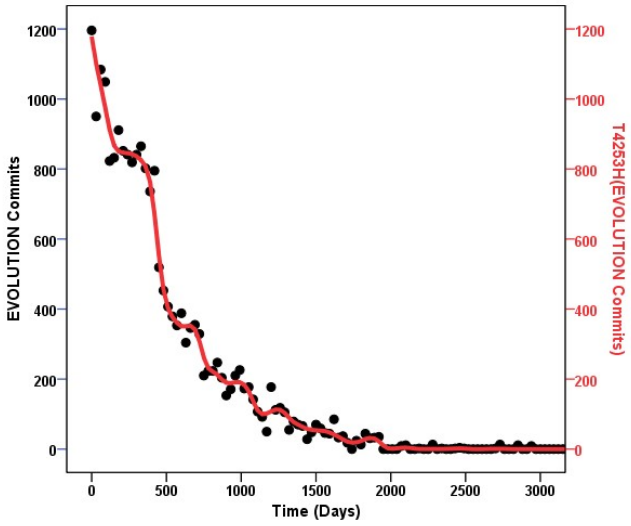
3.12.3 Results

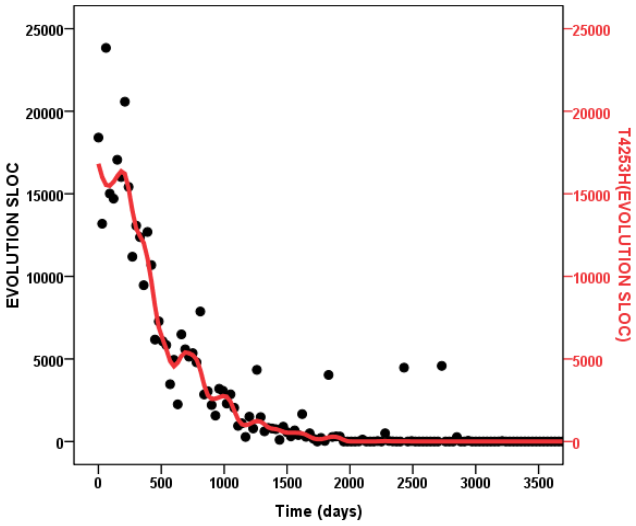
First in Figures 2 (a-h) we present the time distribution of the number of commits and total SLOC *change* (added lines minus deleted lines of uncommented LOC) for each project. The *x*-axis in all plots is the time difference in days between the first commit of any contributor and the date of the current commit. Therefore, it is not *absolute calendar* time for the project but *relative* time of involvement of a contributor with the project. The time bin was taken to be 30 days. All curves represent *average* no of commits and SLOC change over *all* contributors of a project. In each of the following plots a smooth curve is fitted to the data. For the construction of the smooth curve the T4253H smoothing algorithm was used. Specifically, T4253H produces new series by applying a compound data smoother to the original series. The smoother starts with a running median of 4, which is centered by a running median of 2. It then re-smooths these values by applying a running median of 5, a running median of 3, and hanning (running weighted averages). Residuals are computed by subtracting the smoothed series from the original series. This whole process is then repeated on the computed residuals. Finally, the smoothed residuals are added to the smoothed values obtained the first time through the process [3]. In the following plots the original data are presented with the points while the smoothed series is presented with a red line.

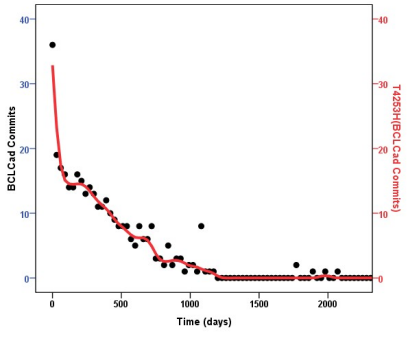
We can clearly see that the number of commits and SLOC change are highest in times close to the first commit of a contributor and gradually decrease to zero. In a few of cases (eg. Python, gtk, Plone) we see sudden bursts of activity in later days of a contributor a fact that signifies a revival of activity of some programmers who have been inactive for some time.

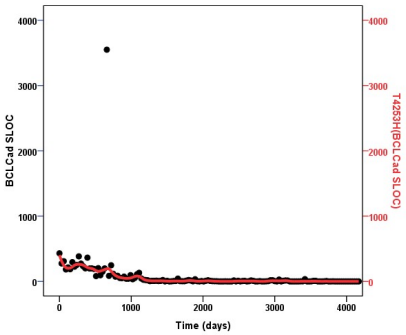
Figure 2: On the left column: Distributions of the number of commits vs time since the first commit. On the right column: distributions of total SLOC change vs time since first commit. For the case of BCLCad, the number of SLOC presents a sudden peak at 660 days, affecting the resolution of the plot. That is why we give the corresponding plot with and without the outlying point.

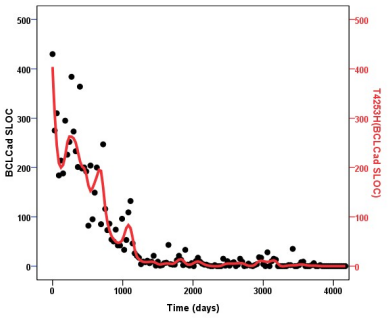


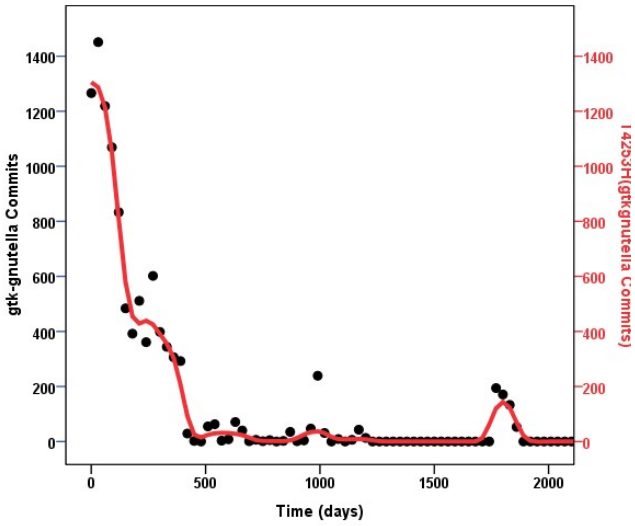

EVOLUTION Commits
T4253H(EVOLUTION Commits)
Time (Days)

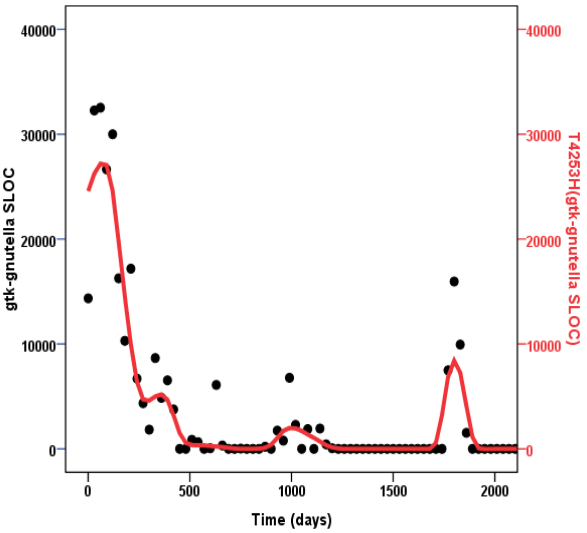

EVOLUTION SLOC
T4253H(EVOLUTION SLOC)
Time (days)

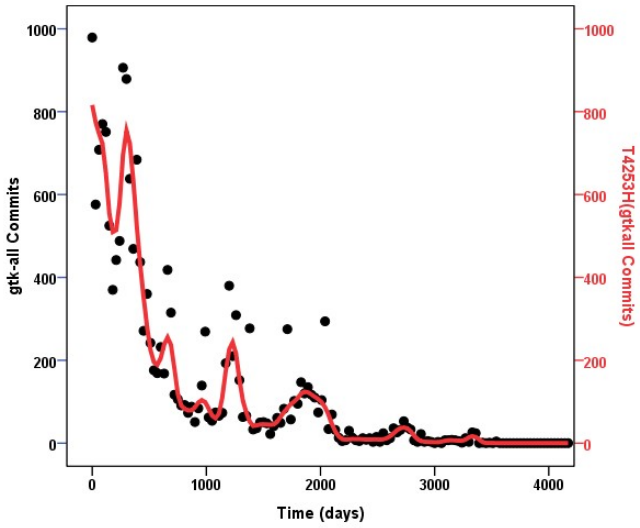

BCLCad Commits
T4253H(BCLCad Commits)
Time (days)

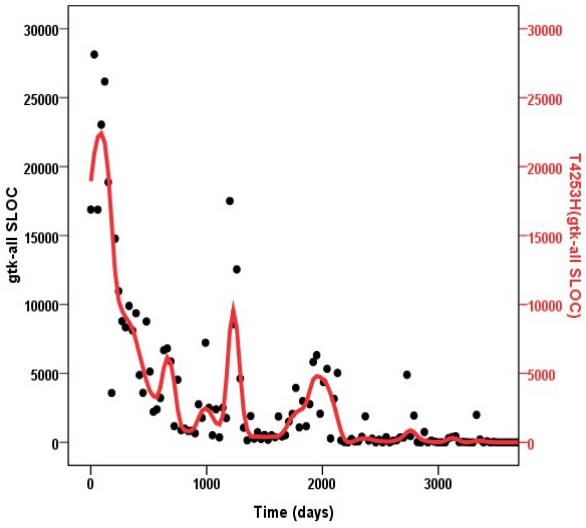

BCLCad SLOC
T4253H(BCLCad SLOC)
Time (days)

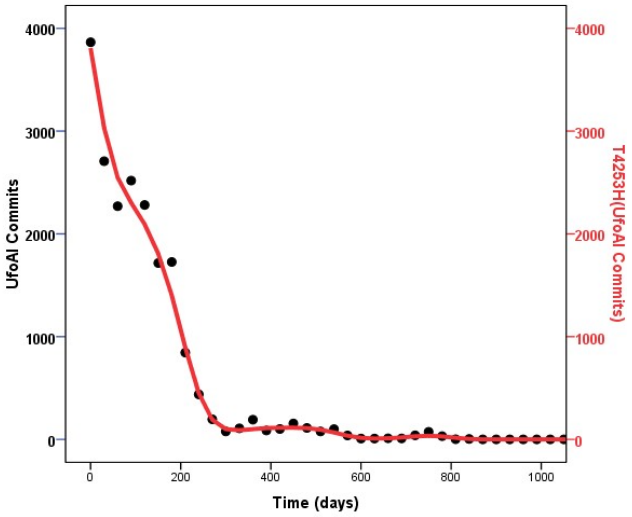

BCLCad SLOC
T4253H(BCLCad SLOC)
Time (days)

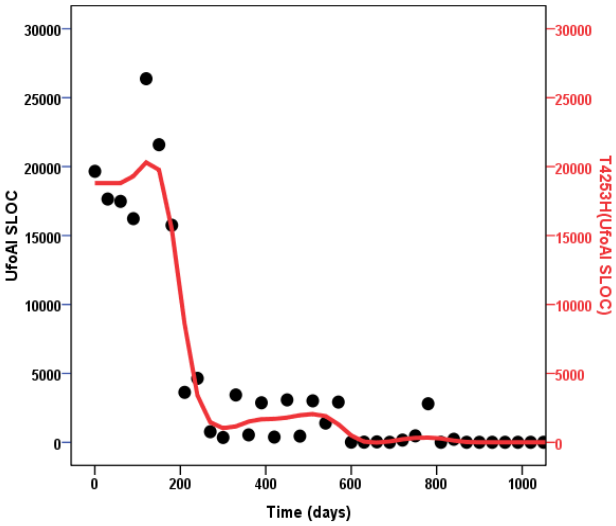

gtk-gnutella Commits
T4253H(gtk-gnutella Commits)
Time (days)

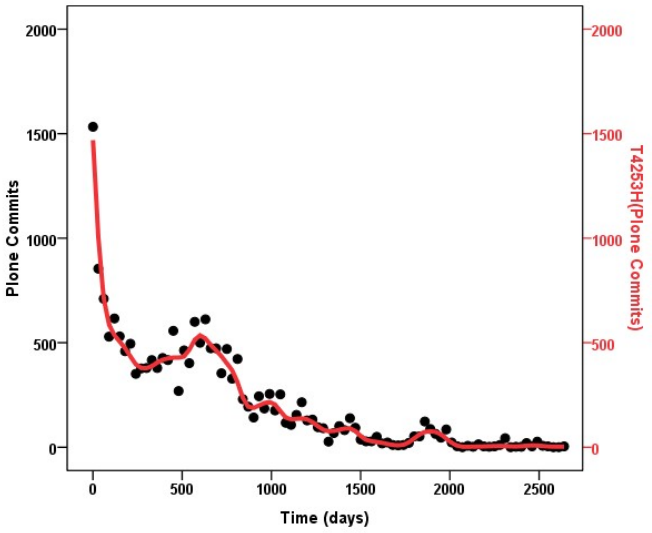

gtk-gnutella SLOC
T4253H(gtk-gnutella SLOC)
Time (days)

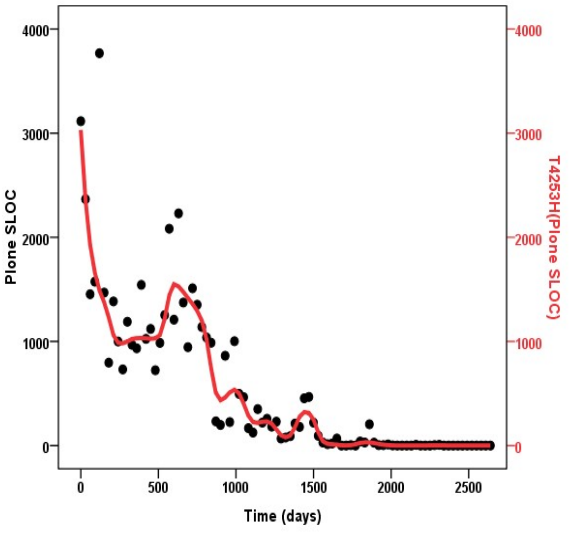

gtk-all Commits
T4253H(gtkall Commits)
Time (days)

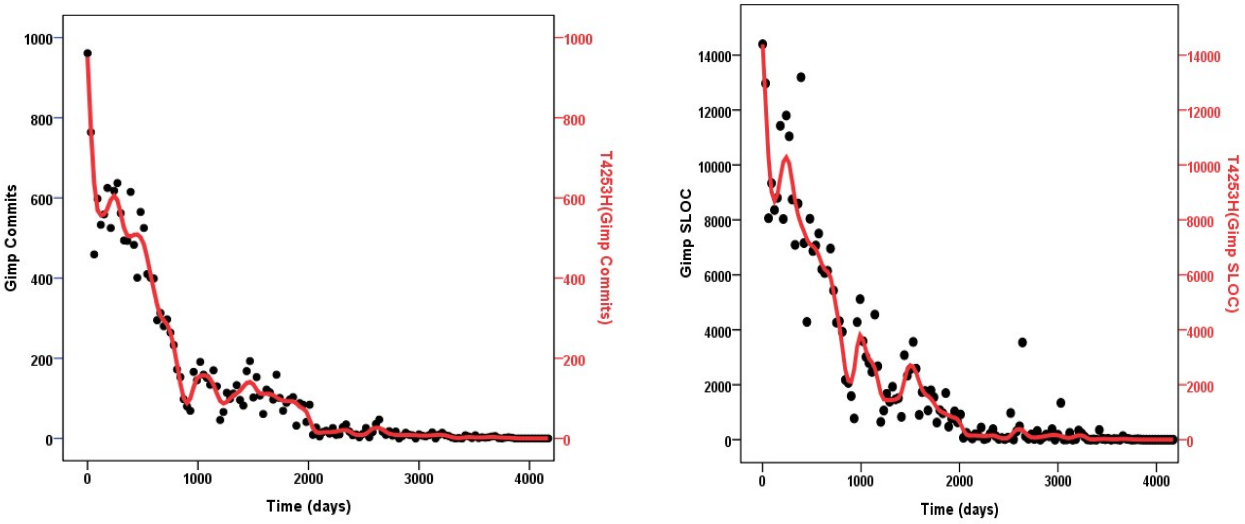

gtk-all SLOC
T4253H(gtk-all SLOC)
Time (days)


UfoAI Commits
T4253H(UfoAI Commits)
Time (days)


UfoAI SLOC
T4253H(UfoAI SLOC)
Time (days)


Plone Commits
T4253H(Plone Commits)
Time (days)

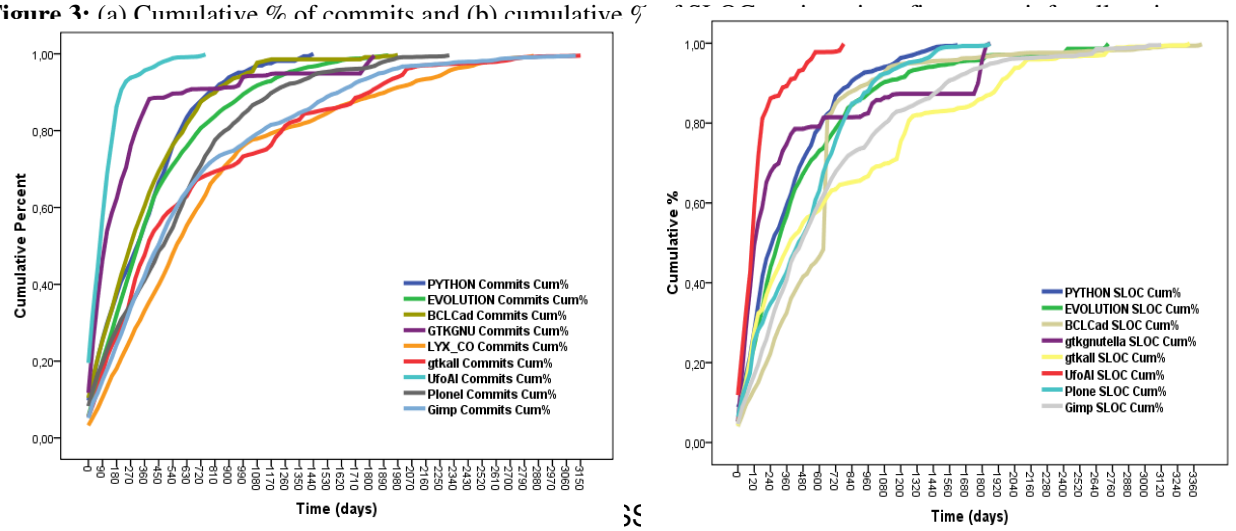

Plone SLOC
T4253H(Plone SLOC)
Time (days)



In all projects the general trend is that a contributor gradually “loses interest” in the project after some time. This is the same as was assumed in Figure 1, although there is no trend of initial increase in productivity followed by a decrease. In this sense, the basic theoretical assumption assumption in the simulation model has to be altered.

Also, the time that a contributor gradually “loses interest” in the project differs significantly among projects. Thus, in an attempt to bring the growth of the projects in a common base, we calculated the maximum number of cumulative Commits and cumulative SLOC reached after some time for all the projects. Then, each cumulative number was divided by this maximum so as to express the growth in terms of percentages. So, each project begins with a 0% percent of commits and SLOC and after some time reaches the 100% where it is stabilized. These percentages are plotted next against the time intervals so as to see the form of the growth. First we present (Figure 3) all cumulative curves together in order to observe the differences in the rate of growth and the total time interval up to 100% (1.00).

Figure 3: (a) Cumulative % of commits and (b) cumulative %



The plots provide useful information, for example we can find easily when the 50% of the total commits or SLOC were reached. See for example the cumulative curve for the Commits of PYTHON. The 50% of the total commits of a particular care reached in about 310 days (see the dashed lines – a horizontal from 0.50 and from the point of intersection with the curve we draw a vertical to see the time). In a sense this information can be viewed as the cumulative probability distribution (CDF) function of the time needed to reach the maximum value (where the project is stabilized).

Next, if we consider the above as cumulative probability distribution (CDF) functions of the time *T* to reach the maximum value of each project, they show a close resemblance with the CDF of the exponential distribution:

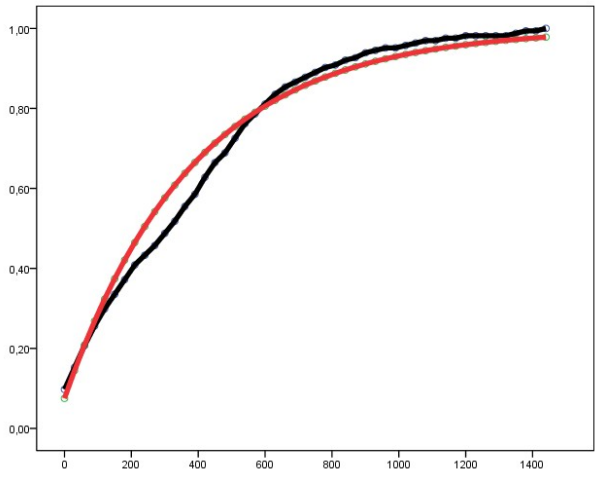
$$F(t;\lambda) = 1 - \exp(-\lambda t), \quad t \geq 0. \qquad (1)$$

The exact shape of the theoretical curve is determined by the parameter λ which is equal to the inverse of the mean of the random variable (i.e. $E(T) = 1/\lambda$).

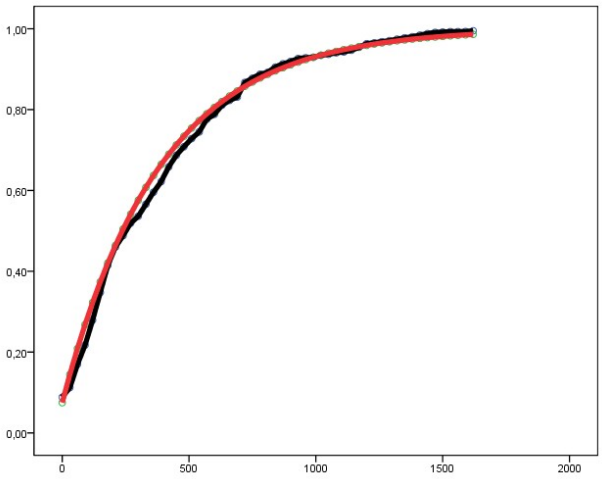
$1/\lambda$ is essentially the characteristic time (averaged over all contributors) relative to the first commit for reduction of contributor productivity to 1/e its initial value.

So, in order to fit the exponential curve to each of the above curves, we had to calculate an estimation of the “mean time”, separately for each project. This is achieved by the following method:

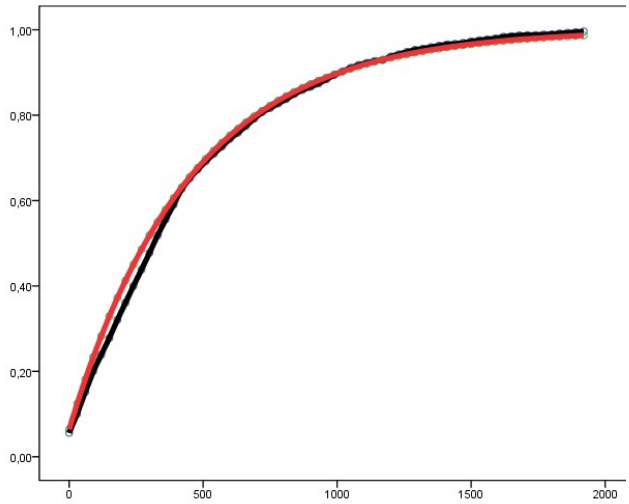
First we note that $F(t;\lambda) = P(T < t)$, so by having in our data (as resulted from the above transformations) the values of $P(T < t_1)$ and $P(T < t_2)$ for consecutive $t_1 < t_2$, we can easily calculate the probability $P(t_1 \leq T \leq t_2) = P(T \leq t_2) - P(T \leq t_1)$. This probability value (within interval) is next multiplied by the center point of the interval $(t_1 + t_2)/2$. All these products are then summed in order to obtain an estimation of the mean which is equal to $1/\lambda$. The estimated value of λ is then used to obtain the curve corresponding to the CDF of the exponential distribution for all the available time values. In figure 4 we present the same curves as above together with the fitted CDF (red line) of the exponential distribution. For each of the curves which are either very good or reasonable approximations of the empirical curves, we also provide the estimated mean, i.e. the value of $1/\lambda$.



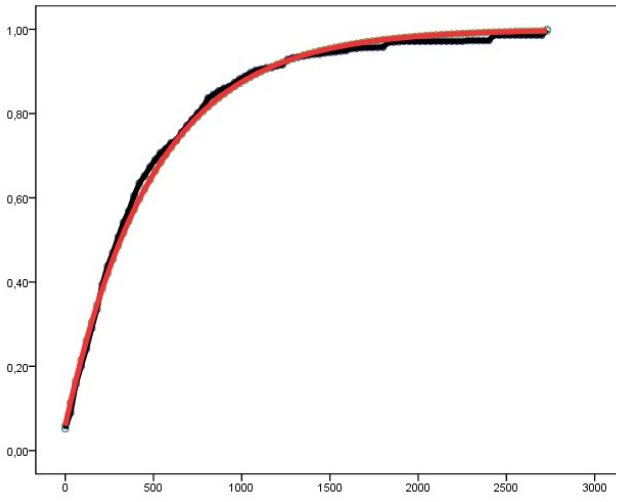
PYTHON Commits: 1/λ=384.15 days



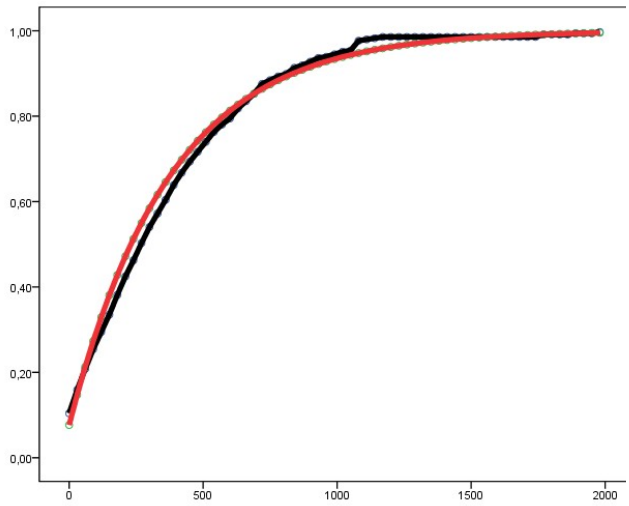
PYTHON SLOC: 1/λ=384.71 days



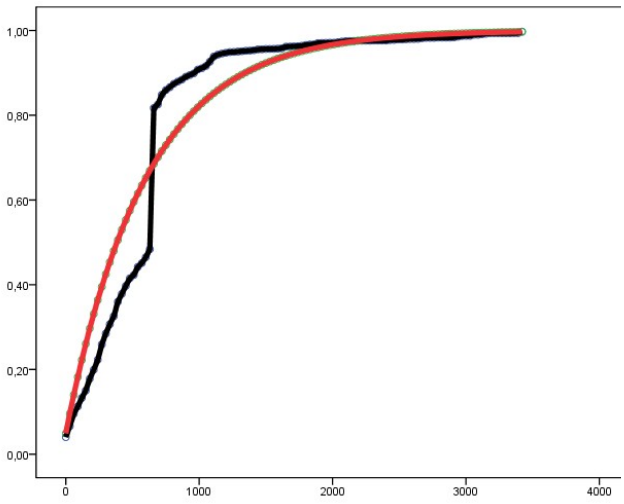
EVOLUTION Commits: 1/λ=451.09 days



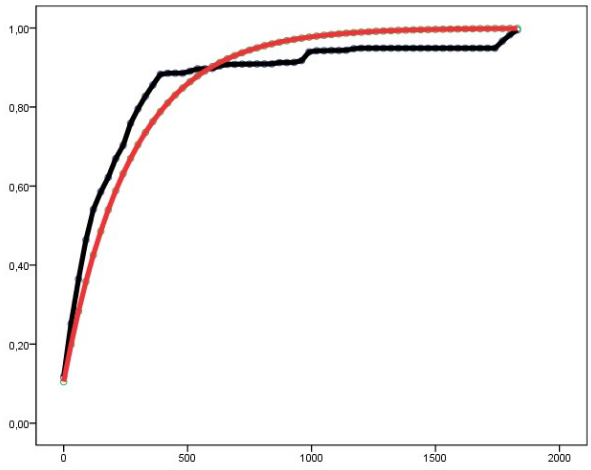
EVOLUTION SLOC: 1/λ=495.75 days



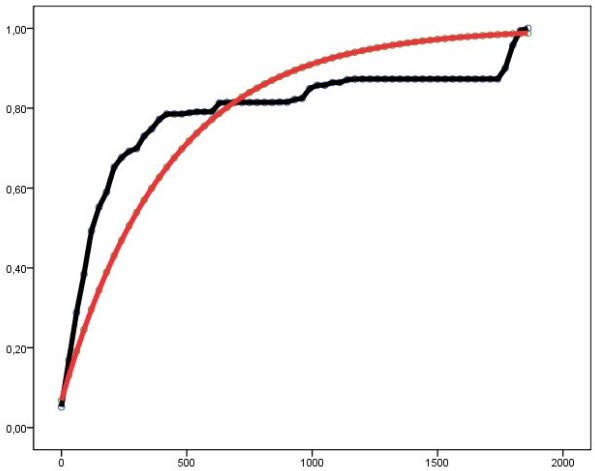
BCLCAD Commits: 1/λ=375.82 days



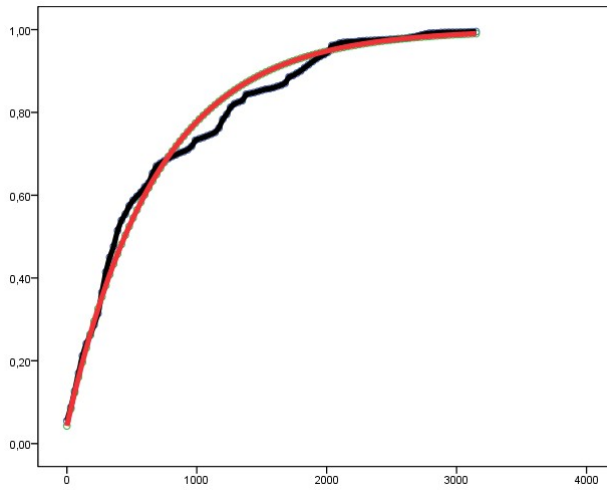
BCLCAD SLOC: 1/λ=596.40 days



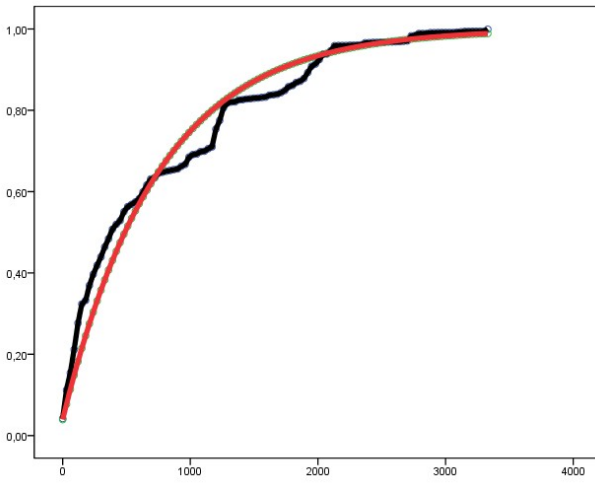
GTKGNUTELLA Commits: 1/λ=270.56 days



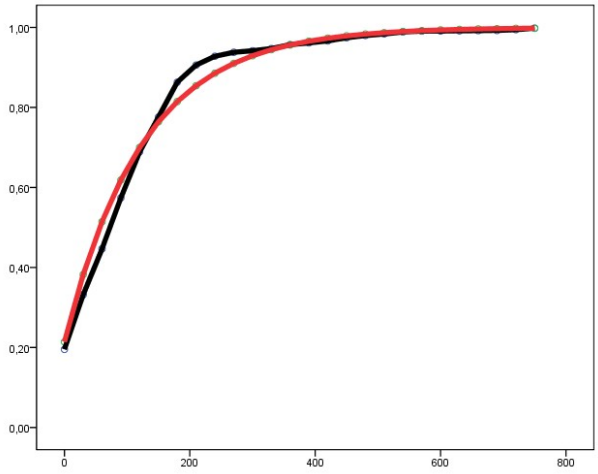
GTKGNUTELLA SLOC: 1/λ=426.30 days



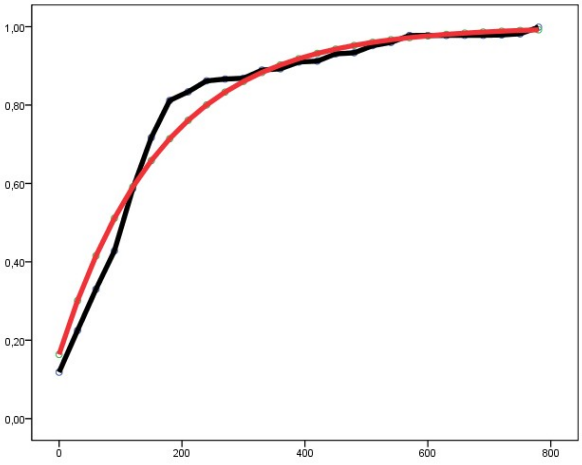
GTKALL Commits: 1/λ=685.50 days



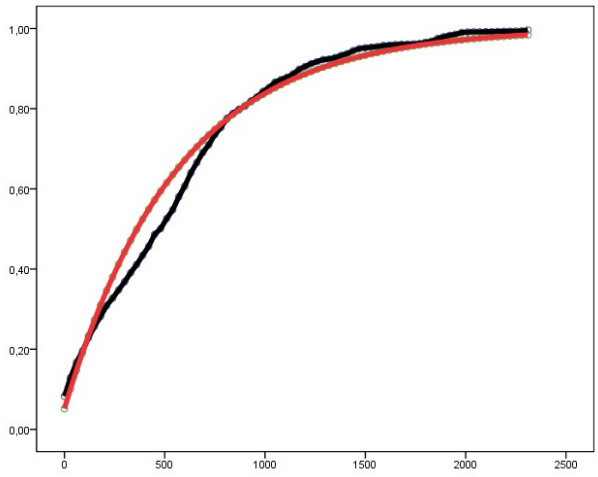
GTKALL SLOC: 1/λ=745.35 days



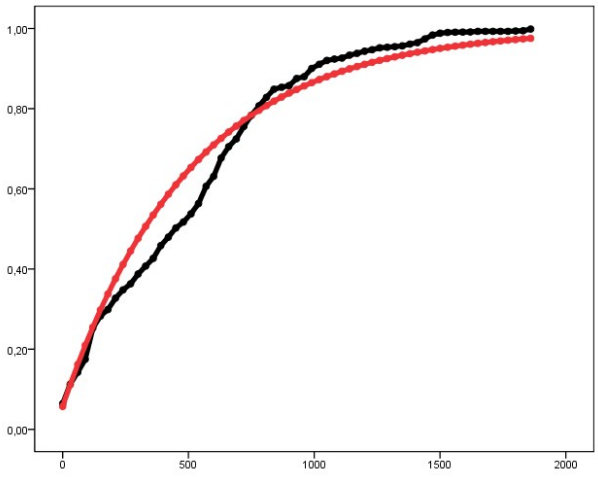
UFOAI Commits: 1/λ=124.56 days



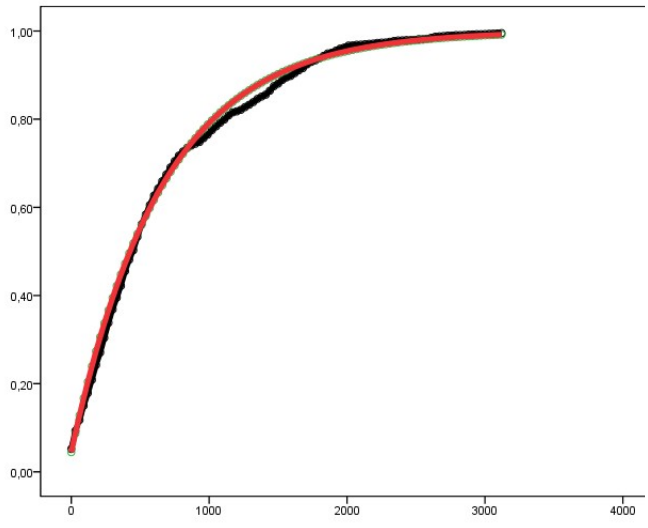
UFOAI SLOC: 1/λ=167.61 days



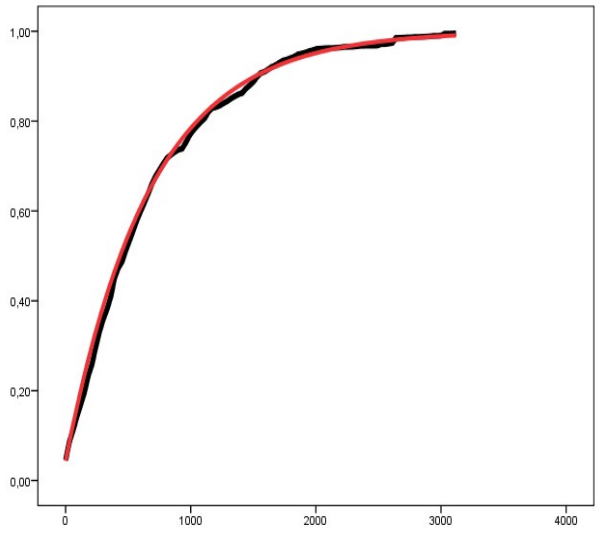
PLONE Commits: 1/λ=565.07 days



PLONE SLOC: 1/λ=509.53 days



GIMP Commits: 1/λ=657.11 days



GIMP SLOC: 1/λ=671.29 days

Figure 4: Cumulative fraction of commits(left column) and SLOC (right column) relative to maximum number of commits and SLOC respectively. Red lines are fits to Equation 1. Characteristic times $1/\lambda$ are also shown.

Therefore, the general conclusion is that the growth of the cumulative number of Commits and SLOC to a maximum point of stability, follows approximately the exponential law. For most project this is a very good approximation with the exception of gtk-gnutella and to a lesser extend Ufo AI

and Ploen which show a more rapid drop of contributor productivity, in the case of gnutella and UfoAI and less rapid in the case of Plone. Each project has its own characteristic time1/λ for Commits and SLOC. In general, the cumulative curves are very well approximated by an exponential distribution with the appropriately estimated λ. Characteristic times as calculated by the cumulative distributions for Commits and SLOC respectively are usually very close for the same project although there are exceptions like the GTKGNUTELLA and the BCLCad. The similarity means that there is a very close correlation between productivity as measured by number of commits and SLOC change. In the case of BCLCad a single commit which resulted in a very large code change at around day 700 caused the deviation from this trend. In gtk-gnutella, on the other hand the deviation was caused by a sharp increase in code production per commit at later times.

It is interesting to compare characteristic times of different projects. Obviously, projects that have run longer would be expected to show larger characteristic times of contributor productivity reduction, although this is not necessarily true: project productivity at later times in project life may be due to “new blood” of contributors that joined at a later phase, or project activity may have dramatically reduced in later times as a whole. In the case that total project activity is maintained at high levels throughout project lifetime, if characteristic times are proportional to project duration, it suggests that the same programmers that joined from the beginning stay active for a time that depends on project life, i.e. most production is based on old programmers. On the contrary, for projects that characteristic times are short relative to its duration (and total activity is relatively constantly high), it means that production is maintained all the more by new contributors and that old ones lose their interest fast.

In order to investigate this question, we show in Table 1 total project durations in days compared with characteristic times obtained both from commit and SLOC curves. In the rightmost two columns, we show characteristic times as percentage of the total project duration. Notice that these percantages are low for some projects (Python, BclCad), medium (UfoAI, Evolution, Gimp) and high (Plone, gtk). For Python for example, although it has the largest duration of all projects examined (~6500 days) it has very low characteristic times for contributor productivity reduction. Also, Python is a project with a total activity that has increased in the past few years. Both the above facts suggest that contributors are faster renewed and “new blood” comes in to continue development. Indeed, if one looks at the individual SVN log of Python, the trend of old programmer IDs being not appearing in later time commit entries is strong. In gtk and Plone, on the other hand, development is more based on old contributors who sustain productivity levels for longer periods relative to project lifetime. See also Figure 5.This can also be verified by looking at SVN logs.

Table 1: Comparison among project durations and characteristic times 1/λ of projects.

Project	**Total project duration (days)**	**1/λ (commits)**	**1/λ (SLOC)**	**[1/λ (commits)] / project duration**	**[1/λ (SLOC)] / project duration**
Pytho	6543	384.15	384.71	5.87%	5.88

n					%
UfoAI	1273	124.56	167.61	9.78%	13.17 %
Plone	2644	565.07	509.53	21.37%	19.27 %
Evolution	4111	451.09	495.75	10.97%	12.06 %
bclCad	9279	375.82	596.4	4.05%	6.43 %
gtk (all)	3235	685.5	745.35	21.19%	23.04 %
Gimp	4156	657.11	671.29	15.81%	16.15 %

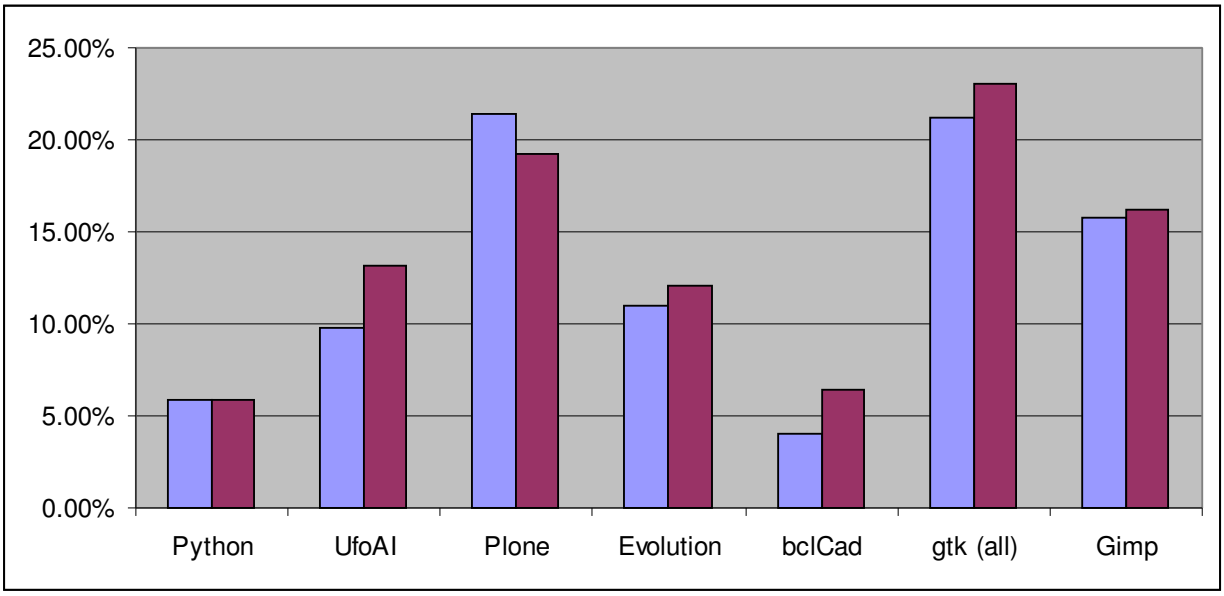
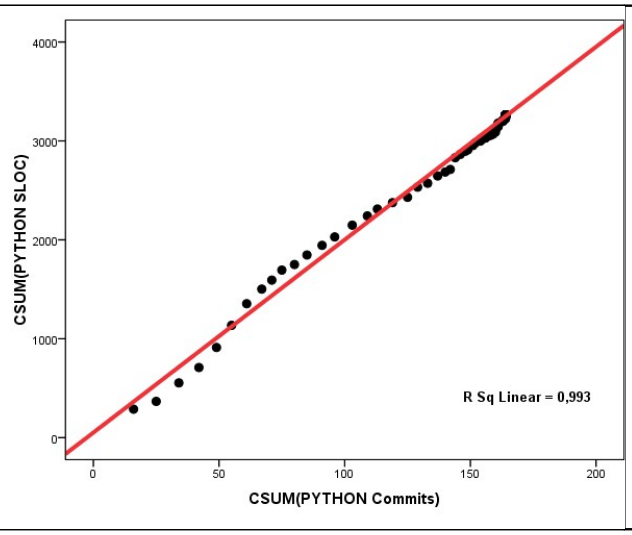


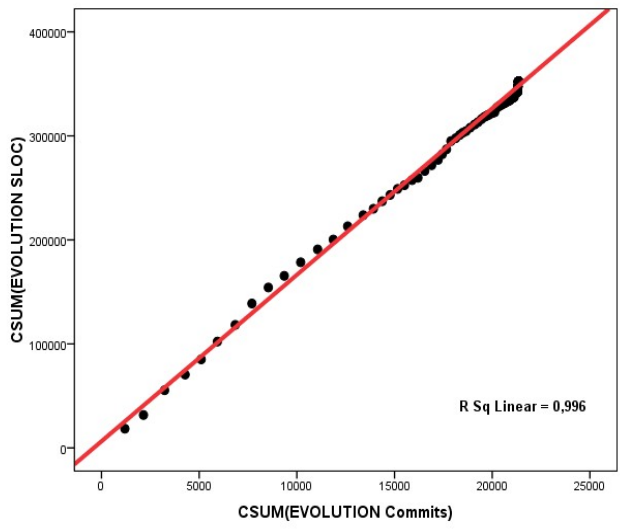
Figure 5: Characteristic times 1/λ as percentages of the total duration of the project. Bars on the left correspond to characteristic times measured from commit curves and bars on the left from SLOC curves.

Finally, in the following plots (Figure 6) we present the correlation between Cumulative Commits and Cumulative SLOC along with an attempt to model their relationship.



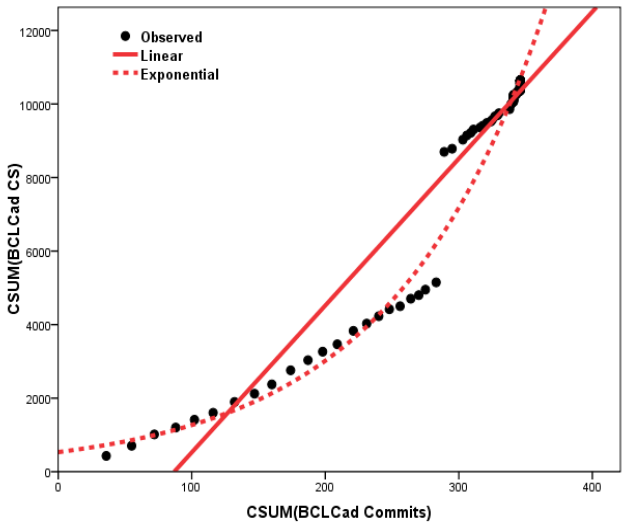
The cumulative Commits and the Cumulative SLOC for the PYTHON project have an almost perfect linear relationship (r-square=0.993). The linear expression that describes their linear relation is

CumSLOC=51.105+19.504*CumCommits



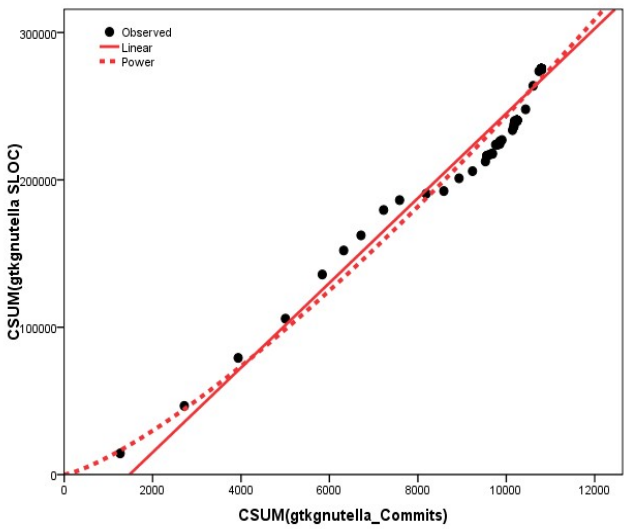
The cumulative Commits and the Cumulative SLOC for the EVOLUTION project have an almost perfect linear relationship (r-square=0.996). The linear expression that describes their linear relation is

CumSLOC=6384.69+16.01*CumCommits



The relation between cumulative Commits and Cumulative SLOC for the BCLCad project is not expressed very efficiently by the linear relationship (although the r-square=0.935) due to a huge jump in the cumulative number of SLOC. The following exponential model (represented by the dashed curve) describes better their relation (r-square=0.981):

CumSLOC=532.74*exp(0.009*CumCommits)

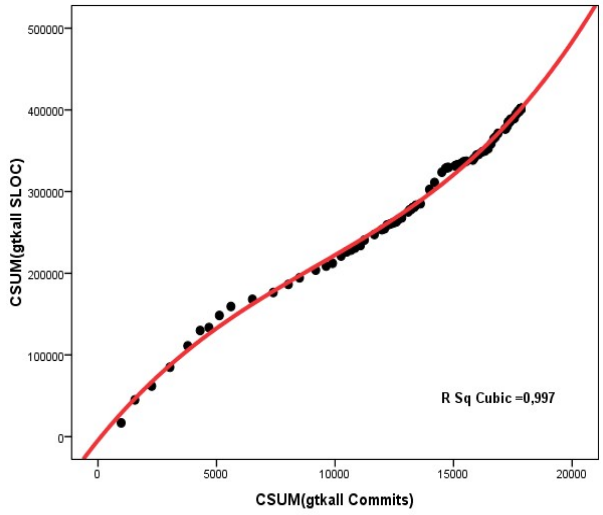


The relation between cumulative Commits and Cumulative SLOC for the GTKGNUTELLA project is expressed quite efficiently by a linear relationship (r-square=0.942) but even better by the following power model (represented by the dashed curve) (r-square=0.982):

CumSLOC=1.458*(1.306**CumCommits)

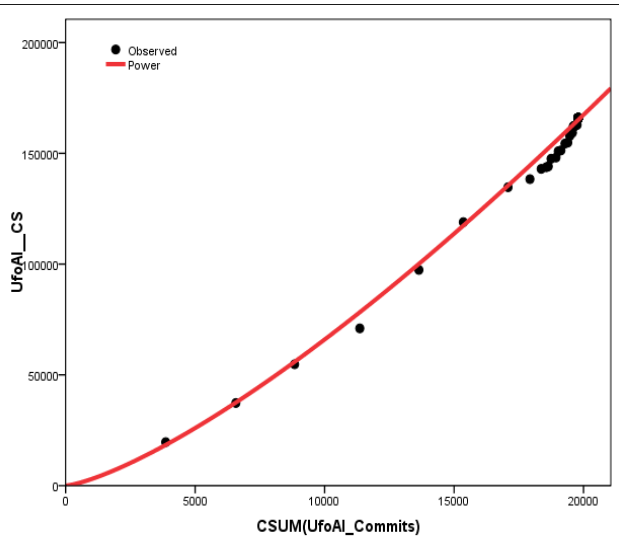
or

$CumSLOC=1.458*1.306^{CumCommits}$



The relation between cumulative Commits and Cumulative SLOC for the GTKALL project is expressed almost perfectly by a cubic relationship (r-square=0.997):

$$CumSLOC = -4518.44 + 35.88*CumCommits - 0.002*CumCommits^2 + 7.45*10^{-8}*CumCommits^3$$

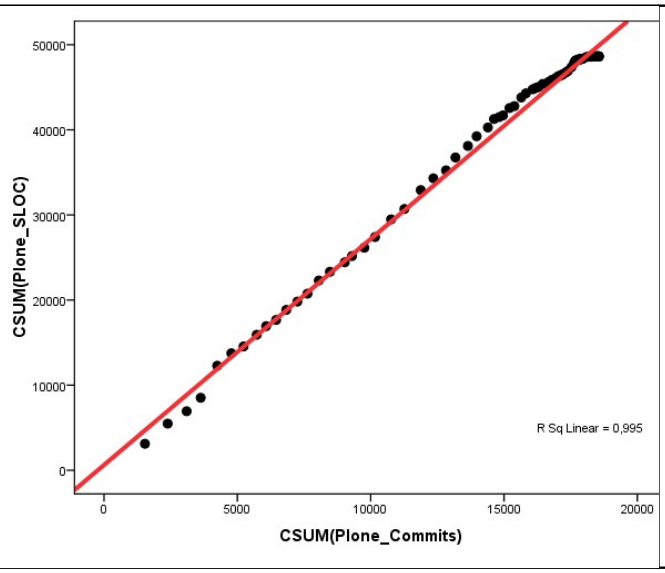


The relation between cumulative Commits and Cumulative SLOC for the UfoAI project is expressed almost perfectly by the following power model (r-square=0.996):

$$CumSLOC = 0.279*(1.343^{CumCommits})$$

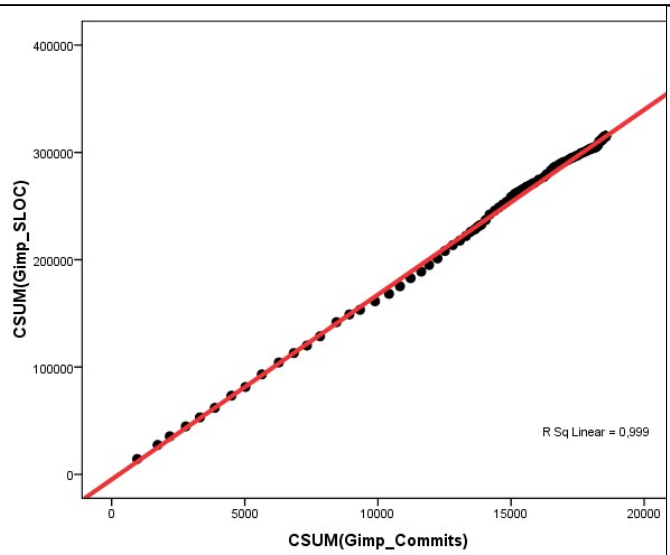
or

$$CumSLOC = 0.279*1.343^{CumCommits}$$



The cumulative Commits and the Cumulative SLOC for the PLONE project have an almost perfect linear relationship (r-square=0.995). The linear expression that describes their linear relation is

CumSLOC=623.23+2.66*CumCommits



The cumulative Commits and the Cumulative SLOC for the GIMP project have an almost perfect linear relationship (r-square=0.999). The linear expression that describes their linear relation is

CumSLOC=-4743.21+17.225*CumCommits

Figure 6: Cumulative SLOC vs cumulative commits. In all projects, except for gtk and bclcad we see an almost linear relationship.

3.12.4 Conclusion

In this study we saw the average dependence of contributor productivity on time since a particular contributor's first commit for several projects after analyzing the SVN logs in

FLOSSMETRICS database. We measure productivity either by number of commits or by SLOC changes to the project source code repository.

The basic conclusions are:

1. Productivity is highest at the first month of a contributor's involvement in the project and gradually decreased to zero at later times. There is no initial increase in productivity, thus the basic assumption made in refs. 1 and 2 is disproved in this sense.

2. There are occasional bursts of productivity, i.e. a contributor may cease to contribute for some time and then come back. This time can be long, even years.

3. There is a very close correlation between productivity as measured in number of commits and uncommented LOC in most projects, but not all. Discrepancies are due to (a few) commits with an exceptionally large amount of code submitted in a single commit, or even a sudden increase of LOC submitted per commit relative to the average for the whole project.

4. There are clear trends in the characteristic times for reduction of contributor productivity among various FLOSS projects: In some projects, these times are small compared to total project duration, a fact which has been interpreted mainly as a faster renewal of the contributor base of these projects, whereas in other projects these times are high, a fact that suggested that development is based on old programmers for a much longer time.

3.12.5 References

- [1] I. Antoniades, I. Stamelos, L. Angelis, G. Bleris. "A Novel Simulation Model for the Development Process of Open-Source Software Projects", *Journal of Software Process Improvement*, Wiley, Special Issue on Process Simulation and Modeling, 7 (3-4) pp. 173-188, (2003)

- [2] I. Antoniades, I. Samoladas, I. Stamelos, L. Angelis, G.L.Bleris, in Eds: S. Koch. "Dynamical Simulation Models of the Open Source Development Process", *Free/Open Source Software Development,* IGI Global, 2005, and in *Global Information Technologies: Concepts, Methodologies, Tools, and Applications*, F. B. Tan, IGI Global, 2008

- [3] Velleman, P. F., and D.C. Hoaglin. "Applications, basics, and computing of exploratoty analysis", Boston Mass: Duxbury Press 1981.

3.13 Substitution Cost Estimation

Note: This study will be formally included in the deliverable “D11.3 – Cost/effort estimation study” from the workpackage “WP11 – Productivity”. More information about this study is described in the related paper “On the Approximation of the Substitution Costs for Free/Libre Open Source Software“ included as annexe to this document.

3.13.1 Introduction

Substitution cost is the monetary value of the effort necessary for implementing a FLOSS application from scratch in a software company. This monetary amount has many potential uses, e.g. it may be used to estimate the gains from reusing a FLOSS component instead of building it from scratch. We have run a pilot study [1] that has helped identifying various problems, limitations and issues related to the data and the model precision. During the pilot study we produced estimation models (see Methodology section) and estimated substitution costs for a subset of FLOSS applications in Debian. The ultimate target of this study is the application of those models on the entire FLOSSMetrics code base.

3.13.2 Methodology

The approach for estimating substitution cost is based on building multiple generic estimation models for the modern software industry and apply them collectively on whatever FLOSS code base we want to estimate. The models should be representative of the modern, global software industry. The models should capture productivity levels that would support our requirement for estimating costs of a typical, average productivity software company. We will use the ISBSG dataset for generating estimation models, taking into account the following:

1. Because of the inherent uncertainty in this estimation context, we should build more than one model. This is a typical precaution in software cost estimation. The precision of the models should be first assessed and then they should applied to the FLOSS projects under study to produce a variety of estimates. Such estimates should then be compared in order to (a) produce the final estimate and (b) assess the overall precision of the calculated substitution cost.

2. The models should be based on some measure of physical size. Functional sizing (e.g. Function point analysis) is not practiced in FLOSS, and it would be infeasible to measure the functional size of thousands of FLOSS projects, given that no automation tools are available for such task at the moment. We chose to use Source Lines of Code as a measure of physical size because they are readily available for FLOSS projects and expected to be retrieved from FLOSSMetrics DB. However limitations of SLOCs are

already known, so we need to take any possible precaution for their use in the model. In particular, we should keep in mind that SLOC measurements may contain a lot of noise.
3. The models should be based on closed source projects similar to those found in FLOSS. In particular, it should be based on projects that are similar to the FLOSS projects for which it will be used to produce substitution costs. Project attributes that define project identity, and potentially affect productivity and therefore cots, are the project application type, development platform, language type, size etc. We should build generic estimation models that take into account these attributes, then, while calculating substitution costs, carefully chosen values to characterize FLOSS projects should be used.

4. The estimation models should produce interval estimates to account for data uncertainty. Given that the models will be calibrated on multi-organizational project data and will be applied on hundreds or thousands of projects developed by communities of volunteer programmers, a range of possible cost values should be produced. Ideally, the estimate for each FLOSS project and for the entire set of FLOSS projects should be a range of values along with a probability distribution.

The major sources of data are ISBSG data and FLOSSMetrics project data, namely SLOC and qualitative project characterisation. ISBSG data are filtered according to their quality, decided by maintainers of ISBSG: only A and B quality rating data are considered. SLOCs are retrieved from FLOSSMetrics database through the use of a script. Qualitative project data are not currently stored in the FLOSSMetrics database. However, due to other partner requests are expected to be available in the database in the next few months. In case this does not happen they will have to be retrieved independently.

3.13.3 Results

The results can be found in the document "D11.3 – Cost/effort estimation study" from the workpackage "WP11 – Productivity".

3.13.4 References

- [1] Kirsten Haaland, Ioannis Stamelos, Rishab Ghosh, Ruediger Glott. "On the Approximation of the Substitution Costs for Free/Libre Open Source Software", submitted to BCI2009

3.14 Productivity

Note: This study will be formally included in the deliverable "D11.2 – Productivity study report" from the workpackage "WP11 – Productivity".

3.14.1 Introduction

Productivity is an economically important and interesting metric, that has been hard to come by for Free/Libre Open Source (FLOSS) programmers and projects. Productivity relates output to input, and thus also provides an indication of efficiency. In traditional software engineering it is the norm to measure how productive software engineers are, mapping input to output, and often implementing appropriate incentive schemes. For FLOSS it is difficult to obtain the data that allows the calculation of productivity. The output of programming activities are mostly readily available through public repositories and software version control data ("CVS"). However, all these available data sources are output measures, and to get the productivity we need to relate the output to the input. Even though the time the changes were checked in to the repository is available, it does not say anything about the amount of time and effort the programmer actually spent on making the changes. Further, it is well known a programmer spends time on various other activities, not all directly related to coding, such as communicating on mailing lists and writing documentations. Therefore, to make estimates of the real effort being the input to the output we can observe in CVS, we link the survey data of time FLOSS developers devote to a specific project for a specific time period to the actual CVS activity in that period.

This high level study is a productivity study, specifically entitled "Estimating the productivity for Open Source projects by calculating productivity estimator function". First step of the study is to obtain the estimate for the individuals having responded to the survey, and develop a spot productivity estimator function based on the survey data combined with the CVS data, and apply this to the projects covered by the survey. Further the analysis outlines the relationship between a developers weekly gross output (in terms of SLOC count) and the total weekly hours developers spent working on their respective projects. Regression analysis and nonparametric measures shows interesting relationship between these two values. However, the relationship becomes even more significant when we considered other activities developers are involved in; for example testing software, reading bug reports..

The objective of this study is to enhance the understanding of open source programmer productivity.

- Develop a spot productivity estimator function based on the survey data combined with the CVS data, and apply the estimator function to the projects covered by the survey.
- Further apply the estimator function to more FLOSSMetric projects, to obtain an estimation of the effort and efficiency of FLOSS programming.

- Provide a methodology for estimating productivity based on developer survey and CVS data.

3.14.2 Methodology

The input data for the productivity study is a survey of committers to open source projects which was conducted in 2006. To participate in the survey the commiters had to make their CVS ID available, in order to unique identify an individual and relate it to the CVS tree of the corresponding project, available through FLOSSMetrics. This enables linking input and output and hence making the productivity estimator.

The questionnaire specifically inquires: "*How much time did you spend in a typical week over the past month contributing (in any way) to this project? ... and how much of this total time was spent coding for this project?*". We assume that an average over a 4 week time period gives a better estimation than only the last week, and it is a short enough time period to be in the recent memory of a committer to make a reliable estimate.

What we specifically can estimate:

- How much time would this one person take for these many total lines of net code, over his lifetime in the project?
- What is the trend of developers
  - a) weekly gross output; that is net SLOC developers contributed to their projects on weekly basis
  - b) weekly coding hours; that is net SLOC developers contributed to their projects on hourly basis.

Methodologically, these values are computed by cross-checking that the developers in CVS are the same individuals who have answered the survey questions.

3.14.3 Analysis and Results

From the 159 developers who completed the survey, we have complete data in the metrics we wanted to measure, for 87 individuals. The metrics are divided into three main categories (this summary analysis introduces 2 of the metrics); *(I) coding dependent* metrics obtained from developers activity logs. These are used as inputs to model developers productivity. (ii) time dependent metrics obtained from the survey. There are only two metrics in this category; how much time developers spend in a typical week over the past month contributing to their project (total weekly hours on thisproject) and how much of that time was spent coding in the same project (coding weekly hours on this project). *, and (ii) combined metrics,* obtained by combing code and time dependent metrics. These variables gives an indication of a developer's activity per unit time. For example, *commitsperhour* is the total number of commits a developer makes in 1hr.

From the code dependent metrics, the mean number of files committed by a developer is 1715.08 (Std. Dev. = 5932.31), the mean commits per developer is 5056.15 (Std. Dev. = 11833.769) and the net lines added per developer (mean = 30525.99; Std. Dev. = 82631.625). At the project level, the mean for the total commits (sum = 18787919) per project is 215953.09 (Std. Dev. = 668345.821). The mean weekly gross output per developer is 316.55 (Std. Dev. = 509.45). In total the developers contributed 27,539.52 lines of code. The maximum number of lines contributed by the most prolific developer was 3013.90 and the minimum was 1.25 lines of code. With the exception net SLOC count per developer per project, all the metrics in this category shows similar distribution,

Total weekly hours developers spent on their project, irrespective of what they are doing, and coding weekly hours developers spent on coding ONLY are the two time dependent variables in our analysis. Descriptive statistics shows that the mean weekly hours per developer is 16.01 (Std. dev. = 17.301) and the mean coding weekly hours per developer is 9.69 (Std. dev. = 9.927). The total number of coding weekly hours is less than the total weekly hours because developers are involved in other non-coding activities which may or may not be related to their projects. Bivariate correlations shows that the times developers spend on either coding or other activities are highly correlated with Pearson r = .862 ($p < 0.000$). However, Kendall's tau_b for the ranked values was a bit lower ($\tau = 743$; $p<0.000$).

In modelling developers productivity, we used regression analysis to help us determine the 'best' parameters for a productivity function. Our linear regression model explains 14.6% (R^2=14.6) in the variability of weekly gross output. We increased the predictability of our regression model by adding the influence of paid developers as a predictor in our linear equation. The resulting model summary in the table belwo shows an improvement in the predictability of the variability in weekly gross output.

Table 8: Improved linear model summary with the inclusion of paid developers effect

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
	R Square Change	F Change	df1	df2	Sig. F Change	R Square Change	F Change	df1	df2
1	.428(a)	.183	.163	1.768	.183	9.399	2	84	.000

a Predictors: *(Constant), paid employee, ln(weekly gross output)*

b Dependent Variable: ln(coding weekly hours)

By incorporating paid developers as a predictors, the linear regression model could explain 18.3% (R^2=0.183) in the variability of weekly gross output. This is a slight improvement over the 14.6%.

This represents a significant but small variability in weekly gross output, hence the productivity of the developers in this respect. Thus, we predict that there are other underlying factors which comes into play when explaining the productivity of FLOSS developers. Factors which we further incorporated in the study include variables like “read / release bug reports”, “*test*

*software", "write code/ patches", "package software", "document software", and "*coordinate project development". The effect of these variables on the productivity of the developers is further explored in deliverable D11.2.

3.14.4 References

1. Banker, R. D., Chang, H., and C. F. Kemerer, 1994, "Evidence on economies of scale in software development," Info. & Softw. Technol., 36, pp 275-282.
2. Banker, R. D. and Kemerer, C. F. 1989, "Scale Economies in New Software Development", IEEE Transactions on Software Engineering, vol. 15 (10), p.1199-1205.
3. Bibi, S., Stamelos, I., and Angelis, L. 2008, "Combining Probabilistic Models for Explanatory Productivity Estimation", Information and Software Technology, Elsevier, 50(7-8), pp. 656-669.
4. Boehm, B. 1981, Software Engineering Economics, Prentice Hall PTR, Upper Saddle River, NJ.
5. David and Shaprio, 2008, "Community-Based Production of Open Source Software: What do we know about the developers who participate?" Available at SSRN: http://ssrn.com/abstract=1286273
6. Dolado, J. 2001, "On the problem of the software cost function". Information & Software Technology, 43(1) pp 61-72.
7. Ghosh et. al. 2006, FLOSSIMPACT: The impact of Free/Libre/Open Source Software on innovation and competitiveness of the European Union, available online at: http://www.flossimpact.eu/
8. Kitchenham, B. 2002 "The question of scale economies in software - why cannot researchers agree?" Info. & Softw. Technol., 44, pp 13-24.
9. Lakhani, Karim R. and Wolf, Robert G.,Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects(September 2003). MIT Sloan Working Paper No. 4425-03.
10. Mockus, A., Weiss, D. M., and Zhang, P. 2003. Understanding and predicting effort in software projects. In Proceedings of the 25th international Conference on Software Engineering (Portland, Oregon, May 03 - 10, 2003). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 274-284.
11. Jørgensen, M., and Shepperd, M., 2007 A Systematic Review of Software Development Cost Estimation Studies, IEEE Transactions on Software Engineering, v.33 n.1, p.33-53.
12. Passing, U. and Shepperd M., 2003: An experiment on software project size and effort estimation. ISESE 2003: 120-131
13. Premraj, R., Shepperd, M., Kitchenham, B. and Fordselius, P. 2005. An empirical analysis of software productivity over time. In METRICS '05: Procs. of the 11th International Software Metrics Symposium, page 37, Como, Italy, IEEE.
14. Scacchi, W. "Understanding software productivity," in Advances in. Software Engineering and Knowledge Engineering, vol. 4, 1995, pp. 37–. 70
15. Scacchi, W. 1995. Understanding and Improving Software Productivity, Institute of Software Research, presentation online available: www.ics.uci.edu/~wscacchi

16. Shepperd, M. 2007. Software project economics: a roadmap. In 2007 Future of Software Engineering (May 23 - 25, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 304-315.

4. Appendixes

This section includes research articles based on the studies described in the previous sections. The idea is to provide some results and useful information about to the researches about the possibilities of FLOSSMetrics database. Please note that these appendixes are distributed in a separate and confidential document due to copyright issues.

The list of papers is the following:

- “Characterization of the evolution of software” (Study: Characterization of the Evolution Dynamics of Software)
- “Evolution of the core team of developers in libre software projects” (Study: Evolution of Core Team Members)
- “What Does It Take to Develop a Million Lines of Open Source Code? ” (Study: Effort)
- Assessing FLOSS Communities. An Experience Report from the QualOSS Project (Study: Quality in Open Source Metrics)
- “A study of the properties of libre software” (Study: Correlation Size and Complexity Metrics)
- "Are FLOSS developers committing to CVS/SVN as much as they are talking in mailing lists? Challenges for Integrating data from Multiple Repositories" (Study: Development vs Communication)
- “On the approximation of the substitution costs for free/libre open source software” (Study: Substitution Cost Estimation)