Repositories with Public Data about Software Development

Jesus M. Gonzalez-Barahona,
Megan Squire, Daniel Izquierdo-Cortazar
Universidad Rey Juan Carlos, Elon University
{jgb,dizquierdo}@gsyc.es, msquire@elon.edu

May 11, 2010

Abstract

Empirical research on software development based on data obtained from project repositories and code forges is increasingly gaining attention in the software engineering research community. The studies in this area typically start by retrieving or monitoring some subset of data found in the repository or forge, and this data is later analyzed to find interesting patterns. However, retrieving information from these locations can be a challenging task. Meta-repositories providing public information about software development are useful tools that can simplify and streamline the research process. Public data repositories that collect and clean the data from other project repositories or code forges can help ensure that research studies are based on good quality data. This paper provides some insight as to how these meta-repositories (sometimes called a "repository of repositories", RoR) of data about open source projects should be used to help researchers. It describes in detail two of the most widely used collections of data about software development: FLOSSmole and FLOSSMetrics.

1 Introduction

Nowadays, most software development projects use a variety of software tools and online systems for coordinating their work. This is true of both proprietary software projects and free, libre, and open source (FLOSS) projects. These tools include centralized web download areas, discussion forums, IRC channels, email mailing lists, source code control systems, bug trackers, wikis, etc [Fogel, 2005]. A project team may choose to manage their own tools, perhaps on their own web site, or the team may also choose to use a suite of tools provided by one of many online code forges. (An example of this could be SourceForge [1], Savannah [2] or RubyForge [3]).

[1] `http://sourceforge.net/`
[2] `http://savannah.gnu.org/`
[3] `http://rubyforge.org/`

Regardless of whether the team chooses to host its own repository or to be part of a code forge, the project tools and systems will leave a detailed trace about the activities and products of the project, which can be analyzed by researchers. Because the end product of a FLOSS project is source code that is freely available, the development for the projects is often done in the open as well. When tools like issue trackers or email mailing lists are used for software development on a FLOSS project, the data they produce is very often left open to the public. The result is a very large amount of publicly-available data about software development. For example, the archives of an email mailing list will serve as a record of the discussions between developers on a project. This will be searchable by anyone seeking a history of how decisions were made on a project. An online issue tracking system can show the list of features requested or bugs reported in a project, as well as the record of how the feature or bug was handled by the development team. The issue tracking software produces logs that are searchable by anyone wanting to know more about this process.

This public record of activity should make it possible for researchers to develop empirical studies based on data retrieved from these systems or, to be more precise, from the code forges and project repositories that support those systems [Dyba et al., 2005], [Kitchenham et al., 2002]. However, researchers often discover that even when the data is publicly available (such as it is with most FLOSS projects), the task is made more complicated by the large size and scope of the project repositories or code forges, and the heterogeneity of the projects being studied [Howison and Crowston, 2004], Making sense of all this data for a research study can be a large challenge, especially for large collections of projects, or for small numbers of projects that have been in existence for a long time.

In this paper, we will outline what kind of data is available from the project repositories or code forges, and what each "repository of repositories" (or RoR) is doing to collect, aggregate, and clean the targeted repository or forge data. We will provide examples from two RoRs, FLOSSMetrics and FLOSSmole, showing what data is available on the original repository or forge, and how that RoR makes this data available to researchers. We also discuss the advantage presented to the researcher for using these RoRs: the researcher does not need to collect data independently. This is an important advantage as it can help the entire research community leverage the work done by others, thus freeing up time and effort to analyze the data rather than to collect it. Finally, we show some example research questions that can be answered using FLOSSMetrics and FLOSSmole data. We conclude our paper with some general observations about the future direction of the RoRs, given the needs of the research community.

2 Data retrieval: the first step

In this section we discuss what data is available and how this data can be organized for ease-of-use by the researcher. The large volume of data in public software repositories is usually comprised of:

- Metadata about software projects. Projects, particularly when they are hosted in development forges, are usually annotated with some metadata which is potentially interesting for researchers. Examples of that metadata are: descriptions of the project, list of developers,

rankings (according to some metrics maintained by the forge), licensing schemas, programming language(s) used, language(s) into which the software has been translated, etc.

- Source code, including metadata about how it changed. Nowadays software projects commonly use a source code management system such as Subversion, git, or CVS. When this source code management system is public, as it is with FLOSS projects, not only all the versions of all the source code files can be obtained, but also metadata about who performed all the changes to that code, and when they did it (sometimes 'why' can be ascertained as well). Metrics can be obtained about all releases of files, and about snapshots of the source code tree. Source code metadata can be used to better understand the actors and artifacts involved in the development of the final products of the project.

- Issue tracking information. Issue tracking systems include not only information about bug reports, and how they are fixed, but also about feature requests, wish lists, recommendations and even comments about the functionality of the software products. In some cases, the issue tracking system is also used as a scheduling tool for the project, or a separate system is used to keep that information.

- Discussions (usually in mailing lists, blogs or forums). Most of the decisions projects are carried though open discussions in many projects. Suggestions, comments and requirements are also subject to threads in these systems.

- Documentation (usually in the form of a wiki or blog). Documentation cover most aspects of how to use the software, as well as descriptions of what features are added in each version, known issues in the software, basic facts about the development team, the motivation of the development team in making the software, and may also explain how interested parties can get involved in the project.

In most cases, historical data for each of these categories is also available, letting researchers reconstruct the state of the project at any given point, and its evolution over time.

It is clear that the activity trail and metadata found at these project repositories and code forges can prove to be very useful for researchers wishing to understand the process of FLOSS development. However, the extraction of useful information out of the forge or repository is not always a simple task. Some of the common problems found by researchers when willing to use information from software repositories are:

- Different systems, with different APIs for accessing the information. For example, and only in the area of source code management systems, there is a good collection of systems used by projects: CVS, Subversion, git, Bazaar, Mercurial (to mention just some of the most used). Each of them has a different API, and requires different means to obtain information. In other domains, the problem is similar, in some cases increased by the fact that only web-based interfaces, designed to be accessed only by browsers, are available, which leads to the need of sophisticated spidering technologies.

- Different representation of the same kind of information. Even if tools for retrieving the data are available, the representation of the data can be different, either semantically, syntactically, or both. For example, the semantics of a commit in CVS, Subversion and git is different enough to need different treatment in most research studies. Another well known example is the representation of a bug report. The bug-fix cycle in different issue tracking systems can be quite different, even between different flavors of the same system, such as is the case of Bugzilla.

- Performance issues, in particular when performing studies on a large number of projects, or on projects that are large themselves. Retrieving the whole history of tens of millions of lines of code from source code repositories, or millions of issues from issue tracking systems, or millions of mailing list messages from every project on a code forge is a difficult and time consuming task. To be productive, researchers can implement checkpoint and rollback procedures, and incremental downloading of data. Many times these have to be engineered with APIs not designed for that. The actual downloads and metrics calculation for a large collection of projects can take months, even with advanced tools.

- Lost data. In some cases, data from a software repository is lost. This is common in cases of transitions to new tools, such as in the recent years as many projects make a migration from older version control systems like CVS to Subversion or other modern decentralized source code management systems. In some cases the project team carefully migrates all the history of its source code. However, in other cases, the old information is only available from the original repository, and over time this data can disappear partially or completely.

- Damage to project infrastructure. In most cases, software development repositories are not designed to provide data to researchers, nor are they optimized to be queried in the way that researchers need. When researchers use the repository for data collection purposes, this can lead to performance problems in the repository infrastructure as it is tasked with answering requests from researchers rather than its primary user, the FLOSS development team. A common example of this might be when a researcher writes a data collection script to quickly spider a whole issue tracking system. There have been times that a code forge has banned access to machines used by a researcher, because the forge administrators perceive repeated research requests as a denial of service attack on its servers.

- Lack of expertise. Researchers may lack expertise in navigating the repositories and code forges, or they may not be able to write the best software needed to efficiently and effectively extract data from code repositories. They are experts in the analysis of data from a certain point of view, but not necessarily on dealing with the many APIs and spidering technologies needed to actually collect that data.

Collection of the data is a very important first step for many research studies, but data collection also requires researchers to work with engineering problems outside of their primary research area. In some cases, this leads to researchers having to devote significant amounts of time and effort to master the data retrieval process, an unnecessary extra burden on their research. In other

cases, a researcher may lack the background or means for gathering the data they really need, so they are forced to use only a fraction of the data they would want. In still other cases, they may reject an entire line of research when they perceive that the data is too difficult to collect. In fact, from anecdotal experience, we think that many researchers, especially those not in the software engineering fields, do not attempt this kind of empirical, quantitative research because, while they may have an interesting research question to answer, it is also really difficult to identify and access the data they need to answer that question.

3 RoRs: Releasing researchers from the burden of data collection

Repositories for researchers (or repositories of repositories, RoRs) with data about software development are the answer that the research community has produced during the last few years to address the aforementioned problems.

These RoRs usually hold data collected from project repositories, and some of them also store some analysis and metrics calculated on the retrieved data. The results (raw data, summary data, and / or analyses) will be stored in a database and accessible to the rest of the research community. In some cases, the database is directly accessible to researchers via a direct query method. In the case of large RoRs, it is also common that database "dumps" are provided. Researchers can download these large data files, and insert the data into their own database or statistical tools. Database dumps can be provided in several formats, including SQL statements (used to re-create a local copy of all or part of the RoR database), or comma-separated value (CSV) files, used for importing the data into spreadsheets or statistical packages. In both cases, a web site serves as the user interface for selecting the data to query or the dump to download.

To be useful to the research community, the databases collected by RoRs have to meet some conditions:

- Accuracy
- Traceability
- Reproducibility
- Completeness (no hiding of information)
- Respect to privacy

In addition, it is important that the retrieval process does not cause performance penalties on the infrastructure of the involved project repositories or code forges.

When using an RoR, the two phases of data retrieval and preparation and the analysis of the retrieved data can be clearly separated and performed by different teams. Researchers interested only in the analysis of data which is already provided by the repositories can benefit from the RoR in several ways.

- No need of data retrieval engineering. Researchers do not need to devote time to sometimes complex programming, dealing with the peculiarities of the infrastructure supporting the repositories and the APIs to access them. They can just download the database dumps, and start from there, being confident that the data was properly downloaded. Additionally, the process followed by the RoR is traceable and reproducible. Researchers are also, in some cases, freed from integrating heterogeneous data from multiple project repositories. For example, researchers interested in the evolution of some particular metric in source code can obtain data from a RoR for many projects using different SCMs. Or, a researcher interested in software licensing can easily see the most popular license types on many different code forges without being limited to a single project or a single forge.

- No need to wait for the data. Downloading the data, and especially for large datasets, is not only a matter of having the appropriate tools: it is also a matter of waiting. In many cases, days, or even months, are needed to obtain information from large collections of repositories. By using RoRs, the only time involved is the download of the database dumps, which usually is quite short compared to getting it from the project repositories.

- Huge amounts of data available. Even investing the time to develop retrieval tools and waiting for the data to be downloaded, it is difficult to obtain data from large collections of software projects, and even more difficult if those projects are in different forges. Worse, after completing the retrieval, the researcher may notice that the set of projects selected was not good for the research goals, or that the data retrieved is incomplete. This will further delay the process of actually analyzing the data. When using a RoR, doing preliminary analysis with large quantities of projects involved is easy, and finding the proper projects for specific studies is much more easy, since their data can be quickly retrieved, and verified for the purposes of the study.

- Easy and quick evaluation of the parameters available for analysis. Software development repositories include a lot of information, and using homegrown tools usually means retrieving only the information that is thought to be needed for some specific study. Time and effort constraints usually result in a no-frills program design, which means that only the specific parameters thought to be needed are retrieved. If later in the research process researchers notice that some important parameter is missing, a new retrieval loop has to be started. Contrariwise, RoRs are usually interested in retrieving as much information as possible. This means that most, if not all, of the parameters are already in the dumps. If the researcher has to extend the range of parameters considered from one iteration of the research study to the next, there is no issue with retrieving new data from the RoR.

- No need of specific agreements with forges. In some cases, there are data points which are difficult or impossible to obtain through the usual API, and specific agreements with forges are needed. For an independent researcher, this is just another barrier to the final results. If the data is obtained from a RoR, these agreements are largely transparent to the researcher. A good example of this case is the Notre Dame agreement with SourceForge, which let researchers use data which would have been difficult to obtain without that agreement.

- Homogeneous data between research teams. The peculiarities of the data retrieval process can make it difficult to compare results between research groups. Usually, different parameters are chosen by the different teams, preprocessing methods are different, and in many cases, even the original project repositories are no longer available. Using dumps from a RoR makes it easier to do these comparisons and reproductions, since the data are labeled, homogenized, and expected to be available for long periods of time.

- Focus on the analysis phase. Most researchers in this domain want to be focused on the analysis phase, which is also where they are most productive. Retrieving the data is just a tedious burden. If they had a resource that provided the data in a reliable and trustworthy format, it is generally preferable for them to just get the data and start analyzing it. This is of paramount importance in the cases of researchers who do not have the time, the resources, or the skills needed to retrieve the data from the projects repositories. Using RoRs, they can just get the data, devote some time to understanding it, and start right away with their analyses.

- Availability of cooked data. Many RoRs maintain not only raw data as obtained form the original repositories, but also 'cooked' or summary data, obtained by performing some kind of analysis on it. For example, in addition to having all the commit records from a SCM repository, a RoR may offer some per-month or per-week metrics of important parameters, based on those, which could be enough for some study on software evolution. Or, a RoR may collect the registration dates for projects on a code forge and summarize these on a graph showing growth in number of projects on a forge over time. These kinds of summaries may simplify the process of learning how to use the RoR, showing what data is available, ultimately making it possible for researchers to jump into their analysis more quickly and with more confidence in the data.

One growing area of interest with the RoRs is that small communities of users of the data in these repositories are starting to emerge. These communities allow for the exchange of experiences between users, or in this case, researchers. In the future, it is expected that these communities will help to speed up the learning process of using the data, and develop their own common expertise about best ways to analyze it.

4 Repositories with data targeted to researchers

There are already several RoRs of potential interest to researchers. In this section, we will present some of the most popular, among those specifically targeted to them.

4.1 FLOSSmole

FLOSSmole is a collection of datasets summarizing known information about free, libre, and open source software (FLOSS) development. This project was started in 2004 and is still active, with approximately 30-50 gigabytes of data added each month. The FLOSSmole project team writes

software to collect most of the data in the collection, but the project also accepts data donations from related project teams, and works collaboratively with teams that collect similar data. Most of the data in FLOSSmole is actually metadata describing various facts about FLOSS development projects housed on software forges such as Sourceforge, Rubyforge, Tigris, or Google Code, or listed in directories such as Freshmeat or the Free Software Foundation directory.

Code forges and directories provide a valuable resource for the software user community to find and download relevant software, but they also provide a wealth of data that can help researchers answer questions about the current state of FLOSS development: how many projects are there, how many people have downloaded the software, what are the programming languages or operating systems that are most common, how are the teams organized, what types of problems are being solved with open source software, what development methods are being used?

FLOSSmole currently collects data from Sourceforge, Freshmeat, Rubyforge, ObjectWeb, the Free Software Foundation, Tigris, Google Code, Github, and Savannah. Currently under development are collectors for Launchpad and enhanced collectors for Google Code, and mailing list collectors for Tigris and Sourceforge. Data is collected from the public web sites of these forges/directories on a monthly or bimonthly basis, depending on the forge in question, and this data is re-released through the FLOSSmole web site in a variety of formats: downloadable text files (tab-delimited for import into Excel or other software), downloadable SQL files (CREATE and INSERT statements suitable for import into a favorite database application), and through a live query tool (free registration required to obtain a username and password).

The data available in the FLOSSmole project, while focused on metadata and facts, is varied and plentiful. There are over 400 GB of data in the main database as of this writing, all of it available for download or via live query. Each dataset is given a unique number indicating which forge the data came from, and date on which the data was collected. There are currently 230 datasets in FLOSSmole, spanning over five years. It is possible to track the evolution of a project over time by identifying the name of a project and tracing it through the years of datasets. It is also possible to analyze the rate of growth in new projects and the death or waning of existing projects, languages, or development paradigms.

4.2 FLOSSMetrics

FLOSSMetrics is a FP6 project [Herraiz et al., 2009] funded by the European Commission. Its main goal is to provide a large scale database with information from FLOSS projects. It became fully functional in 2009.

Data in FLOSSMetrics comes from the publicly available data sources found in FLOSS projects, such as source code management systems, mailing lists, bug tracking systems and source code releases. The data is retrieved, stores in an SQL database, and analyzed. Therefore, in addition to having access to the raw data, the analysis performed on it may help to better understand the world of libre software development, to obtain factual data to improve the software development process and to identify interesting practices which could be used in other contexts.

Currently there are more than 2,800 projects analyzed, although the figure is continuously increasing. In the case of the source code management system, there are around 2,000 projects with data, around 1,400 issue tracking systems analyzed and around 600 mailing list repositories.

In FLOSSMetrics, the information is retrieved automatically, thanks to the Blackbird [4] system developed for it, which integrates some mining tools such as:

- CVSAnalY: stores in a database (MySQL or SQLite) the parsing of a log file from several SCMs such as CVS, Subversion and Git.
- Mailing List Stats: parses information from mailing lists in mbox format and stores it in a MySQL database.
- Bicho: retrieves information from Bugzilla BTS (so far, it works with KDE, GNOME and Apache communities) and also the SourceForge tracker and stores it in a MySQL database.
- CMetrics, CCCC, PyMetrics, PerlMetrics: offer complexity metrics for several programming languages.
- SLOCCOunt: developed by David Wheeler, provides basic information related to number of SLOC and a basic COCOMO model.

All the results of the retrieval and analysis processes are published in the Melquiades website [5], which offers, for each project, several kinds of information:

- Description: summary description of the project.
- Results: general overview to the data available for the project, including the primary databases obtained with the different mining tools aforementioned.
- Resources: URLs where the data sources were found.
- Charts: set of statistics and charts about the given project, for all the set of data sources analyzed.
- Quality Indicators: based on the quality model proposed by the QualOSS [6] project.
- Dumps: each of the databases stored by FLOSSMetrics for this project.

4.3 Comparing FLOSSMole and FLOSSMetrics

As seen, the two presented RoRs are complementary since FLOSSMetrics is mainly focused on developer's activity (source code management, mailing lists and bug tracking systems) while FLOSSMole is a true RoR in the sense that this project collects any data provided by third parties. Thus, FLOSSMole is not so interested in having all the possible data from a given project, but offers data from several different repositories and even not only focused on developer's activity. On the other hand, FLOSSMetrics is deepening more in each analyzed project offering metrics and charts for

[4] http://git.libresoft.es/blackbird/
[5] http://melquiades.flossmetrics.org
[6] http://qualoss.eu

each of them. Thus, the difference between the two as one of breadth versus depth. FLOSSMetrics is depth. FLOSSmole is breadth.

In addition, FLOSSMole accepts donations of data from other projects where it will obtain a reference entry (id number) and release (up to once per month) like any other data set. Hence, if there is a donation of analysis across repositories this would be accepted inside the FLOSSMole data sets. On the other hand, FLOSSMetrics has specific tools to retrieve data and to create metrics per project or per FLOSS ecosystem. This complicates the fact of analyzing cross repositories and this is something not expected in this RoR.

In other words, FLOSSMole would accept the analysis of FLOSSMetrics, but the analysis done in FLOSSMole does not fit at all with the FLOSSMetrics data.

4.4 Some other RoRs

Some others RoRs currently in production worth mentioning are (the list does not pretend to be exhaustive):

- SRDA. The Sourceforge Research Data Archive (SRDA) [Antwerp and Madey, 2008] located at University of Notre Dame has several years worth of information donated directly from Sourceforge. Data is available to university researchers who sign a license agreement. SRDA has an ongoing cooperative relationship with FLOSSmole to share a community mailing list, since both teams collect Sourceforge data and can benefit from each other's knowledge about that code forge.

- PROMISE Software Engineering Repository [Sayyad Shirabad and Menzies, 2005] This repository has several example data sets from the time period 2004-2006 covering some different aspects of software engineering research questions, such data sets used to train defect prediction or cost estimation algorithms.

5 Some examples

In this section we develop two simple case studies for two of the RoRs: FLOSSmole and FLOSSMetrics. We show how research questions can be answered easily with the data sets from these RoRs.

5.1 Example of FLOSSmole usage

One reason that researchers like to study FLOSS projects is that they are curious as to whether the FLOSS way of developing software will result in better software (where "better" is defined variously, depending on the research team and question). A university undergraduate student recently asked the question of whether there is a Cope's Rule for software development. He was referring to the "rule" of evolutionary biology that states that animals will evolve to be larger over time. (In biology, this "rule" is not as much a rule as it is a raging debate which is far beyond the scope

of this paper, but I point interested readers to [Gould, 1997] and [Alroy, 1998] to start finding more information.) Using data in FLOSSmole, this student would like to show whether (1) over a particular time, and (2) in particular code forges, (3) project team sizes and (4) did or did not grow larger, and by what margins did this occur.

To answer this question, the student will identify one or more of the code forges in the FLOSSmole database, and he will extract the list of projects on the forge for each time we collected data. For each data collection, and for each project on that collection, he will be interested in the project registration date (start of each project) and the project team size. He will track the team size for each project as it grows or shrinks (or stays the same) over time. He will then be able to add other interesting questions, such as comparing change in team size to change in size of the code base. Since FLOSSmole does not collect metrics about size of code base, he will either have to collect this data and donate it to FLOSSmole or use one of the other RoRs that does collect this data (such as FLOSSMetrics).

The specifics of gathering enough data to get started answering a question like this are fairly easy in FLOSSmole. Within the FLOSSmole database, each data collection from a particular forge on a particular date is given a unique number. We call these data sources. Examples: data source 204 is the December, 2009 collection from the forge Savannah, and data source 205 is the December, 2009 collection from the forge GitHub. Data source 206 is the February 2010 collection from Sourceforge.

The student will need to identify which data sources (dates and forges) he is interested in. To see all data sources, he will go to our database and run the query:

```
SELECT datasource_id, forge_id, friendly_name
FROM datasources;
```

This information about the various dates and forges for data sources is also available on the FLOSSmole web site.

Next, the student will need to identify which forges look promising, and how far back he can go for data in our RoR about that forge. Example, if he is interested in Sourceforge but wants to know how many times we have collected there (more times means more data):

```
SELECT d.datasource_id, d.forge_id, f.forge_abbr, d.friendly_name
FROM datasources d INNER JOIN forges f ON d.forge_id= f.forge_id
WHERE f.forge_abbr = 'SF';
```

Depending on which forge(s) he chooses, he will need to identify the correct location of the developer count data for that forge. For Sourceforge, the developer count can most reliably be found by counting the developers per project per data source:

For each datasource_id, he will run:

```
SELECT dp.proj_unixname, count(dp.dev_loginname)
FROM developer_projects dp
WHERE dp.datasource_id = x
GROUP BY 1;
```

This will yield a count of developers listed for each project in that collection. Then, these counts can be aggregated by project over time into a new table on the student's own machine. Example:

Project Name: abc Dev Count in January 2009 collection: 0 Dev Count in February 2009 collection: 2 Dev Count in March 2009 collection: 4

Project Name: mno Dev Count in January 2009 collection: 10 Dev Count in February 2009 collection: 10 Dev Count in February 2009 collection: 11

etc.

He can then analyze whether the projects seem to grow over time or not. He can modify his original question to take into account growth of the source code base of a project in addition to the developers working on the project, or any other growth metric that he may think would be fruitful.

5.2 Example of FLOSSMetrics usage

One of the common research topics in libre software is how to deal with the fact that communities are lead by volunteers and check their real involvement in terms of effort and evolution of the source code [Scacchi, 2006]. This is still work in progress, but using the FLOSSMetrics database some initial questions are quickly resolved.

Not having access to the FLOSSMetrics database makes the process of retrieving data in some cases too complex. Most of the information that could be retrieved to undertake these analysis is found in the SCM by means of the log system. The relational database provides a way to easily obtain answers for some questions.

For instance, two questions that are easily answered using the FLOSSMetrics database are those related to the evolution of the developers team involved in the project and the number of files which are being touched by those committers.

Then, we are able to match, for a given period, the number of committers with the number of files per type of file. This provides a quick first glimpse about the number of people and quantity of work done during its whole life. [7]

5.2.1 Understanding the FLOSSMetrics website and Databases

All the databases offered by the FLOSSMetrics project can be found at the Melquiades website [8]. Thus, if we are interested in a given project, this is enough to look for it using the searching box available. This will show a new web site with a set of dumps ordered by date and by type of data source.

For this evolution analysis, we just need the CVSAnalY database. Specifically with the next tables:

- scmlog: Every commit in the repository is represented by a record in this table. Main fields are:

[7] More queries can be found at the Melquiades Wiki hosted by the FLOSSMetrics project: `http://melquiades.flossmetrics.org/wiki/doku.php?id=queries`

[8] `http://melquiades.flossmetrics.org/`

  - rev: This is the revision identifier in the repository. It is unique in every repository.
  - date: The date when the commit was done.
  - committer_id: Committer identifier, that is, the identifier in the database of the person who did the commit.
  - author_id: Author identifier. In some cases, the source control management system makes a difference between the person who committed and the real author of that change.
  - message: The commit message written down by the committer.

- action: This table contains the different actions performed in every commit. In systems like CVS, where a commit matches with a file, there is only one record for each commit. On the other hand, most of the "modern" systems, several files could be committed in each commit, what is represented as several rows in this table per each revision.
  - type: This is a single character which represents a type of action: added (A), modified (M), deleted (D), renamed (V), copied (C) or replaced (R).
  - commit_id: This is an identifier of the commit where the action was performed. This is a foreign key that references the id field of scmlog table.
  - file_id: Identified for each of the files handled in a given revision.

5.2.2 Number of distinct committers per unit of time:

This metric shows the number of active committers month. The FLOSSMetrics database provides a table with each committer who has committed at least one change to the repository. Thus, it is only necessary to take all the committers who have committed during each month:

```
SELECT date_format(s.date, '%Y') myyear,
       date_format(s.date, '%m') mymonth,
       COUNT(distinct s.committer_id) committer_author
FROM scmlog s
GROUP BY date_format(s.date,'%Y%m');
```

5.2.3 Number of Files per Type and per Unit of Time

This metric aims to show the number of files touched by a given unit of time (month) and the kind of file. This could show, for instance, months with a high activity in terms of translation for a given project (for instance due to that a release is close to that date). It is calculated by retrieving the files and their type month by month and showing the results ordered by year and month.

```
SELECT   a.type,
         date_format(s.date, '%Y') myyear,
         date_format(s.date, '%m') mymonth,
```

```
COUNT    (distinct a.file_id) numfiles
FROM     scmlog s,
         actions a
WHERE    s.id=a.commit_id
GROUP BY a.type,
         date_format(s.date,'%Y%m');
```

6 Conclusions

RoRs represent an important linking piece between the copious amounts of raw data available about FLOSS projects and the researchers who want to advance science and technology by applying that data to answer their own research questions. In this paper we have presented several desired attributes of FLOSS repositories of repositories, and some advantages that RoRs convey to the researchers when they are not tasked with collecting their own data. We have also outlined the features and contributions of FLOSSmole and FLOSSMetrics, two of the RoRs in active development, explaining how to use these two RoRs by giving real-world examples of usage.

Each RoR has its own peculiarities which may create a wide range of solutions to work with each of them. An example of this is the differences between the FLOSSmole and the FLOSSMetrics repositories: while the first one was initially thought as a place to store data from the projects hosted in SourceForge, nowadays, it offers information from other RoRs or data sources (each one with a different database schema). In the case of FLOSSMetrics, it is focused on the analysis of the commonly found data sources, offering information with a similar schema for all of them, but avoiding some other data sources such as those aforementioned. Thus, both offer a set of data, interesting for researchers, but the methodology used to obtain data for each of them is totally different.

In the future, it is expected that both FLOSSmole and FLOSSMetrics, and some other RoRs, provide increasingly more data, and of better quality. It is also expected that researchers will increasingly use them to avoid the cumbersome phases of data retrieval, at the same time being able of performing large scale studies. However, only time will say whether they will become a usual tool for the researchers in the different aspects of FLOSS development.

7 Acknowledgements

The work of Jesus M. Gonzalez-Barahona and Daniel Izquierdo has been funded in part by the European Commission (QUALIPSO project, FP6-IST-034763) and by the Spanish Government (project SobreSalto, TIN2007-66172).

The work of Megan Squire has been funded in part by the U.S. National Science Foundation (CRI/CRD 07-08767).

References

[Alroy, 1998] Alroy, J. (1998). Cope's rule and the dynamics of body mass evolution in north american fossil mammals. *Science*, 280(5364):731–734.

[Antwerp and Madey, 2008] Antwerp, M. V. and Madey, G. (2008). Advances in the sourceforge research data archive. In *WoPDaSD*.

[Dyba et al., 2005] Dyba, T., Kitchenham, B., and Jorgensen, M. (2005). Evidence-based software engineering for practitioners. *IEEE Software*, 22(1):58–65.

[Fogel, 2005] Fogel, K. (2005). *Producing Open Source Software: How to Run a Successful Free Software Project*. O'Reilly Media, Inc.

[Gould, 1997] Gould, S. (1997). Cope's rule as psychological artefact. *Nature*, 385.

[Herraiz et al., 2009] Herraiz, I., Izquierdo-Cortazar, D., and Rivas-Hernández, F. (2009). Flossmetrics: Free/libre/open source software metrics. In *CSMR*, pages 281–284.

[Howison and Crowston, 2004] Howison, J. and Crowston, K. (2004). The perils and pitfalls of mining SourceForge. In *Proceedings of the International Workshop on Mining Software Repositories*, pages 7–11, Edinburg, Scotland, UK.

[Kitchenham et al., 2002] Kitchenham, B., Pfleeger, S., Pickard, L., Jones, P., Hoaglin, D., El Emam, K., and Rosenberg, J. (2002). Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, pages 721–734.

[Robles et al., 2008] Robles, G., Gonzalez-Barahona, J. M., Izquierdo-Cortazar, D., and Herraiz, I. (2008). Tools for the study of the usual data sources found in libre software projects. *International Journal on Open Source Software and Processes*, 1(1).

[Sayyad Shirabad and Menzies, 2005] Sayyad Shirabad, J. and Menzies, T. (2005). The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada.

[Scacchi, 2006] Scacchi, W. (2006). Understanding open source software evolution. In Nazim H. Madhavji, Juan Fernandez-Ramil, D. E. P., editor, *Software Evolution and Feedback: Theory and Practice*, pages 181–206. Wiley.