Warehousing and Studying Open Source Versioning Metadata

Matthew Van Antwerp and Greg Madey

University of Notre Dame
{mvanantw,gmadey}@cse.nd.edu

Abstract. In this paper, we describe the downloading and warehousing of Open Source Software (OSS) versioning metadata from SourceForge, BerliOS Developer, and GNU Savannah. This data enables and supports research in areas such as software engineering, open source phenomena, social network analysis, data mining, and project management. This newly-formed database containing Concurrent Versions System (CVS) and Subversion (SVN) metadata offers new research opportunities for large-scale OSS development analysis. The CVS and SVN data is juxtaposed with the SourceForge.net Research Data Archive [5] for the purpose of performing more powerful and interesting queries. We also present an initial statistical analysis of some of the most active projects.

1 Introduction

Versioning programs have been in use by open source software projects for many decades. Publicly available logs offer a development trail ripe for individual and comparative studies. In this paper, we describe the downloading and warehousing of such data. We also present some preliminary data analysis. The process is similar to that done in [2] which described an approach to populating a database with version control and bug tracking system data for individual project study. At Notre Dame, Jin Xu also took an individual project approach to retrieving and studying projects on SourceForge [6]. Xu built a similar retrieval framework however for web pages to gather project statistics.

2 SourceForge.net Data

Most of SourceForge's data is stored in a back-end database. The actual source code is stored in a Concurrent Versions System (CVS) or Subversion (SVN) repository. The data stored there includes who is making a change to the code, how the new version of the code differs from the most recent version, the number of removed and new lines of code, a revision number, a comment, and a timestamp. The entire history of a project can be reviewed by walking chronologically through one or more CVS or SVN logs. The logs tell us what changes a project has undergone, when those changes took place, and by whom. We have recently obtained CVS and SVN metadata (everything except the actual code) and built another database that is juxtaposed with the back-end database. This data is available for scholarly research at http://srda.cse.nd.edu .

P. Ågerfalk et al. (Eds.): OSS 2010, IFIP AICT 319, pp. 413–418, 2010.

3 Concurrent Versions System

Concurrent Versions System (CVS) is software developed for software version control allowing simultaneous use by multiple users. It is built upon the Revision Control System (RCS) software, which takes care of individual file versioning [4]. CVS provides a layer of abstraction allowing for concurrent access to a particular RCS file with intelligent conflict mediation. CVS also groups files together into a logical entity (a project) and allows tagging of particular file revisions as a logical snapshot (a project release, for example) [1].

We downloaded log data from all projects on SourceForge.net that use CVS for version management and also allow anonymous CVS access. To do so, the following CVS command was used:

```
cvs -d:pserver:anonymous@PROJECT.cvs.sourceforge.net:/cvsroot/PROJECT rlog .
```

Using `rlog` instead of `log` allows the process to run without having to run a time and space-consuming `checkout` command. The dot (.) at the end is used in place of a module name to indicate that log data is being requested for all modules.

Similarly, we downloaded CVS metadata for the projects on the open source hosting platform BerliOS Developer and GNU Savannah.

```
cvs -d:pserver:anonymous@cvs.savannah.gnu.org:/sources/PROJECT rlog .
cvs -d:pserver:anonymous@cvs.berlios.de:/cvsroot/PROJECT rlog .
```

For obtaining SVN data, the following commands were used:

```
svn log --verbose http://PROJECT.svn.sourceforge.net/svnroot/PROJECT
svn log --verbose svn://svn.berlios.de/PROJECT
svn log --verbose svn://svn.savannah.gnu.org/PROJECT
```

4 Download Process

The method employed was to make one serial line of requests on multiple machines. Due to the lack of physical machines at our immediate disposal, and the ease with which they can be set up, virtual machines were employed. In addition, if a machine were about to make a request to the same CVS server it just contacted, a stall time was employed. When this occurs, the virtual machine (VM) would sleep for 5 seconds before making the next request.

4.1 Job Distribution

We wrote a central server process to handle distributing jobs to each VM. This process spawned a process for each VM which would submit a project name that the VM would download and then return a signal for one of the following: 1) success, 2) initiation failure, or 3) progress failure. Upon return, the handler process would then submit a new project name to the VM it handles and the cycle continues. The information was tracked in a database on the central server.

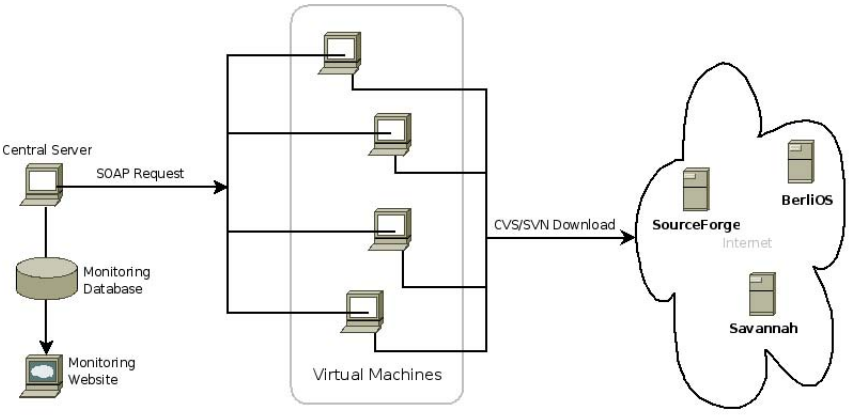

Fig. 1. Diagram of the job distribution and download process. This can be easily modified to retrieve other publicly available data or rsync the code instead of just retrieving the metadata.

The database contained the name of the project, the name of the server (VM) the job was deployed to, the timestamp of the submission, the returned signal, the timestamp of the returned signal, and if applicable, the number of lines in the downloaded CVS log. In order to monitor the download progress and be aware of potential problems, a web frontend using AJAX was deployed to monitor the database. A schematic of this process is shown in figure 1.

SourceForge.net CVS data was obtained over the span of about 7 days. SourceForge SVN data took about 2 days to download. Obtaining CVS and SVN data for BerliOS and Savannah took about 48 hours total. The number of projects successfully downloaded from each site is shown in table 1.

Any changes made to projects since the log files were downloaded are obviously not present in our database. Therefore, continuous updates to the data are necessary, a data warehousing issue brought up in [3]. Two aspects of the database and the log files make this relatively simple to do. CVS contains a filter option to only return log information after a specified time. For each project, we can search in our database to find the most recent timestamp, and then use that as the range specifier and only new updates will be returned. SVN allows a user to specify a range of revisions when running the log command. In this case, we can simply retrieve the number of the latest revision and download all revisions since that one.

Table 1. Number of logs downloaded from each hosting site, classified by versioning software

Hosting Site	**CVS**	**SVN**
SourceForge	103869	24416
BerliOS	1252	1718
Savannah	1775	8

5 The Database

Due to space constraints, details on the log parsing and database design are omitted. ER diagrams for the CVS and SVN database are provided in figure 2.

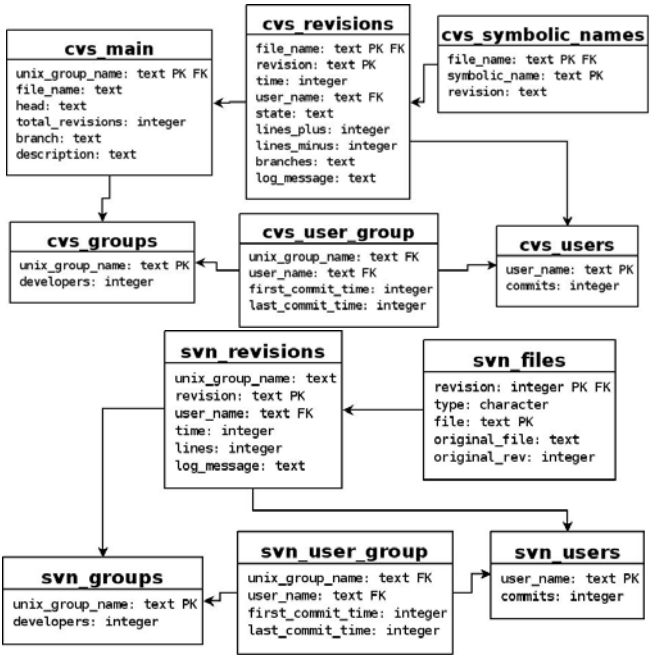

Fig. 2. Entity Relation (ER) diagrams for the new CVS and SVN database

6 Data Analysis

In this section, we present some quantitative data on gcc and emacs, two extremely mature and long-lived open source projects from GNU Savannah.

The GNU Savannah hosted project gcc is the GNU C compiler. The CVS log begins in 1988 and had 345,723 file commits up until November 2005 when the project was transferred elsewhere. Certain months had nearly 10,000 commits. Nearly 300 people have contributed to the project in its 20 year history. The CVS log file for gcc was the largest of all projects that were downloaded, with a size of about 1.5 GB. The graph is found in figure 3.

Another mature project hosted on Savannah is emacs, the popular editor. 200 users have contributed since its initial CVS checkin in 1985. The most active months had over 2500 file commits, with a total of 122,254 commits over all time. The information is graphed in figure 4.

While these are relatively simple quantitative statistics, right away there are interesting portions that warrant further investigation. Both gcc and emacs have a slight, but noticeable lull in activity before a sustained increase in activity.

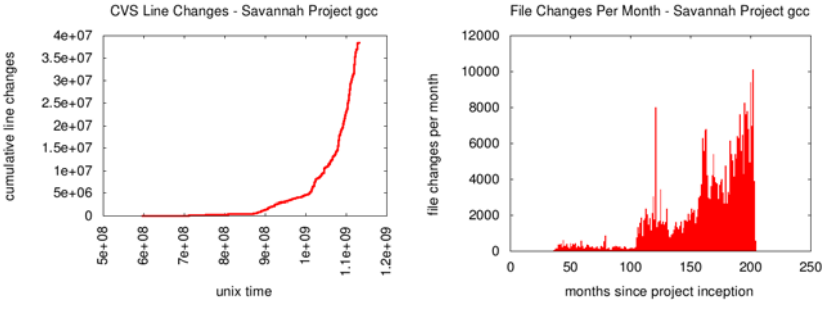

Fig. 3. Savannah project gcc

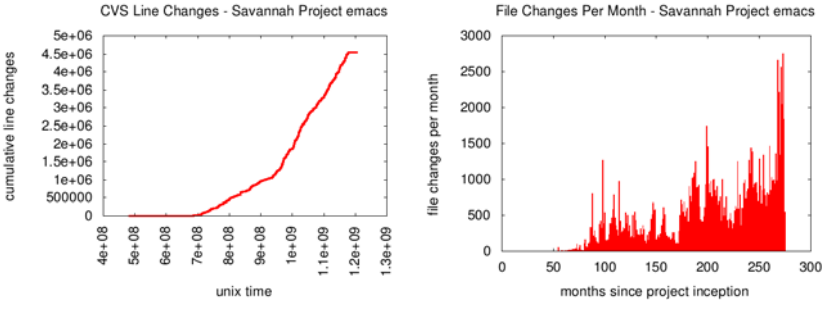

Fig. 4. Savannah project emacs

Comparing the cumulative line changes graphs for the projects, the lull occurs for both approximately mid-2001, roughly coinciding the dot-com bubble burst. Another trend noticed in some of the younger and less mature projects was a distinct pattern in the cumulative line changes graphs. These graphs often showed a period of positive acceleration, then an inflection point, then negative acceleration. This would seem to indicate an increasing number of additions to the software, a peak activity period, followed by a level of code maturity where most of the fixes are minor (patches and bug fixes). The patterns in development activity, and comparison of these patterns across different projects will be examined more thoroughly in a future publication.

7 Conclusions

This large data set offers a multitude of open source software and social networking research opportunities. We can learn about project development trends and group similar projects together by development similarity. We can examine contribution trends by individual coders. We can see how they migrate from project to project and how the amount and types of contributions differ over

time. This site has the potential to become a very important and valuable research hub for researchers of various fields. It is likely that many of the users of our SourceForge Research Data Archive will benefit from the CVS and SVN database and site features.

Acknowledgments

Research reported in the paper was supported in part by the National Science Foundation's CISE IIS-Digital Society & Technology program under Grant ISS-0222829 and by the National Science Foundation's CISE Computing Research Infrastructure program under Grant CNS-0751120.

References

1. Per Cederqvist. Version management with cvs (2002)
2. Fischer, M., Pinzger, M., Gall, H.: Populating a release history database from version control and bug tracking systems. In: Proceedings of the International Conference on Software Maintenance, pp. 23–32. IEEE Computer Society Press, Los Alamitos (2003)
3. Rundensteiner, E.A., Koeller, A., Zhang, X.: Maintaining data warehouses over changing information sources. Commun. ACM 43(6), 57–62 (2000)
4. Tichy, W.F.: Rcs—a system for version control. Softw. Pract. Exper. 15(7), 637–654 (1985)
5. Van Antwerp, M., Madey, G.: Advances in the sourceforge research data archive. In: Workshop on Public Data about Software Development (WoPDaSD) at The 4th International Conference on Open Source Systems, Milan, Italy (2008)
6. Xu, J., Huang, Y., Madey, G.: A research support system framework for web datamining research: Workshop on applications, products and services of web-based support systems. In: The Joint International Conference on Web Intelligence (2003 IEEE/WIC) and Intelligent Agent Technology, Halifax, Canada, October 2003, pp. 37–41 (2003)