STUDYING OPEN SOURCE VERSIONING METADATA

A Thesis

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Master of Science in Computer Science and Engineering

by

Matthew Van Antwerp

Greg Madey, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana
April 2009
STUDYING OPEN SOURCE VERSIONING METADATA

Abstract

by

Matthew Van Antwerp

The SourceForge Research Data Archive (SRDA) is a collection of Open Source Software (OSS) data and resources. Over 200 researchers worldwide use the archive for research in many fields. SourceForge provides us with monthly data dumps mirroring their back-end database, but their versioning metadata is not provided. OSS projects have used versioning programs such as Concurrent Versioning System (CVS) for many decades. Publicly available versioning logs offer a development trail ripe for individual and comparative studies. We describe the downloading and warehousing of such data from SourceForge, BerliOS, and GNU Savannah and the interface and resources we offer for browsing and studying the data. We also present some preliminary data analysis and outline some interesting possibilities for future research that this data provides. This thesis focuses on OSS versioning metadata as well as the recent developments of SRDA.
This thesis is dedicated to my advisor, Dr. Madey, whose patience has far exceeded what was reasonable for anyone to ask of him.
CONTENTS

FIGURES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . vi

TABLES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . viii

ACKNOWLEDGMENTS . . . . . . . . . . . . . . . . . . . . . . . . . . . . ix

CHAPTER 1: INTRODUCTION . . . . . . . . . . . . . . . . . . . . . . . . 1
1.1 OSS Background . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
1.2 OSS Hosting Websites . . . . . . . . . . . . . . . . . . . . . . . . . . 2
1.2.1 Other Open Source Hosting Sites . . . . . . . . . . . . . . . . . 2
1.3 OSS Studies . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.4 OSS Research Collaboratories . . . . . . . . . . . . . . . . . . . . . . 3
1.4.1 SourceForge Research Data Archive . . . . . . . . . . . . . . . . 4
1.5 SourceForge.net Data in SRDA . . . . . . . . . . . . . . . . . . . . . 4

CHAPTER 2: ADVANCES IN THE SOURCEFORGE RESEARCH DATA
ARCHIVE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.2 Other OSS Research Websites . . . . . . . . . . . . . . . . . . . . . . 7
2.3 Data Access . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
2.4 Usage Statistics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
2.5 Documentation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
2.6 Additional Data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
2.6.1 CVS and SVN Research . . . . . . . . . . . . . . . . . . . . . . . 12
2.6.2 User and Project Data and Connections . . . . . . . . . . . . . . 14
2.6.3 Other Open Source Hosting Sites . . . . . . . . . . . . . . . . . 15
2.7 Administrative Concerns . . . . . . . . . . . . . . . . . . . . . . . . . 15
2.8 Features in Development . . . . . . . . . . . . . . . . . . . . . . . . . 16
2.8.1 Stored Queries . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
2.8.2 Automated Graph Production . . . . . . . . . . . . . . . . . . . . 17
2.8.3 Web Service Improvement . . . . . . . . . . . . . . . . . . . . . . 20
2.9 Planned Features . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
2.10 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
APPENDIX C: SOURCEFORGE.NET DATABASE SUBLICENSE TERMS AND CONDITIONS .................. 72

BIBLIOGRAPHY  .......................................................... 75
FIGURES

2.1 The query form page. The autocompletion feature is displayed. . . . . 8
2.2 The query history page. Queries are color-coded by section of the SQL statement. .......................................................... 9
2.3 The schema browser. Autocompletion is also shown here. ............ 10
2.4 Query and download statistics since the beginning of 2006. .......... 11
2.5 Entity-relation diagram for new CVS data. ............................ 13
2.6 Entity-relation diagram for new SVN data. ............................ 14
2.7 Cumulative line changes over time for the GNU C Compiler. The first tick is Nov 1985 and each tick is approximately 3 years and 2 months. 1.2e9 is Jan 2008. ........................................... 18
2.8 Cumulative line changes over time for the emacs editor. The first tick is Sep 1982 and each tick is approximately 3 years and 2 months. . . 19

3.1 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. ......................... 29
3.2 Berlios project cp2k .................................................. 49
3.3 Savannah project gcc ................................................ 50
3.4 Savannah project emacs ............................................ 51
3.5 Savannah project libc ............................................. 52
3.6 Savannah user rms ............................................... 53
3.7 Savannah user drepper ......................................... 54
3.8 Savannah user rms on project gcc ................................ 55

A.1 A user is presented with a simple query form shown above. ........ 60
A.2 A list of users who have contributed to that project is displayed along with a list of distinct commits (MRs) and the cumulative number of line changes for that MR. All displayed information is a link to further details about either that person’s work on this project or the view of all files and comments made for a particular MR. 

A.3 A graph similar to the above is automatically created for the queried project along with additional descriptive information displayed below it.

A.4 This graph is also automatically created. Below that is a list of all files in the file tree. Each file name is a link to the full CVS history of that particular file.

A.5 The full history of a user on a project is displayed. This historical information is that which appears in the CVS logs, however most information is linked so that further data can be browsed regarding a particular MR or file.

A.6 The full revision history is displayed for a particular file. The data is linked similarly to the previous figure.

A.7 A modification request is displayed. 4 files were committed simultaneously by one user. Such a group of simultaneous commits cannot be displayed by cvs tools alone, but is often of interest to researchers.

B.1 A researcher may start at the schema browser to find information on a particular schema and table.

B.2 A user can type in a schema or click on a listed schema to get to this page which lists all the tables in that schema.

B.3 A user can then click on a particular table to get detailed information on the fields in that table.

B.4 Once a user knows what table and field(s) to query, they can enter that query on the query form page shown above.

B.5 While the query is executing, the above page will be displayed.

B.6 When the query is finished, a message will be displayed indicating such and a link to the results is provided.

B.7 A user has a choice between xml and text results. Shown above is a sample xml result set displayed in a browser.
TABLES

3.1 NUMBER OF LOGS DOWNLOADED FROM EACH HOSTING SITE, CLASSIFIED BY VERSIONING SOFTWARE . . . . . . . . . . 31
ACKNOWLEDGMENTS

The material presented in this thesis was supported in part by grants from 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. Y. Gao and S. Christley contributed to early versions of the SourceForge Research Data Archive (SRDA) and J. Xu was an early tester and user.
CHAPTER 1

INTRODUCTION

Open Source Software (OSS) development phenomena have been studied for years, perhaps the most important piece of early OSS literature being Raymond’s *The Cathedral and the Bazaar* [48]. Raymond makes generalizations about OSS and numerous studies since then have analyzed data in light of his statements. Many other quantitative and qualitative studies have been conducted in recent years. In this thesis, the focus is on OSS versioning metadata and development behaviors as well as the SourceForge Research Data Archive (SRDA), an important data resource for OSS researchers worldwide.

1.1 OSS Background

OSS has been around in spirit for decades and was more concretely defined by OSS licenses. The most prominent of these is the GNU Public License (GPL). This license, first published in 1989 [29], ensures rights to access source code, modify, and redistribute software released under the license. If a developer chooses to modify and redistribute GPL-licensed code, he must release that software under the GPL also. In this way, it is a viral, or copyleft license. This nature ensures GPL code cannot legally be included in proprietary software. There are numerous open source licenses, including the less restrictive Berkeley Software Distribution (BSD) license, which allows code to be included in any other kind of software, open source or proprietary.
The Open Source Initiative (OSI) approves open source licenses (although there are a few licenses considered open source which are not OSI-approved) [45]. Today, open source software is prevalent. The Apache web server, the Linux operating system, PHP, Perl, PostgreSQL, Sendmail, and Firefox are all examples of open source software used widely and daily.

1.2 OSS Hosting Websites

SourceForge.net is the world’s largest Open Source Software (OSS) hosting website, boasting over 2 million users and over 160,000 projects at time of writing [54]. SourceForge offers services for open source developers to contact other developers, join existing projects, start their own open source project, and manage all aspects of their project. Software Configuration Management (SCM) resources are also provided. This includes Concurrent Versions System (CVS) and Subversion (SVN), which will be discussed in great detail in this thesis.

1.2.1 Other Open Source Hosting Sites

While SourceForge is the largest OSS development website, BerliOS Developer and GNU Savannah are some other popular OSS hosting websites. BerliOS is a German website with about 1500 users and nearly 3200 projects [4]. GNU Savannah is another hosting site that started when the SourceForge project itself was relicensed as proprietary software [30]. While SourceForge has over 160,000 projects, these small and potentially tight-knit hosting sites offer another rich set of research data. GNU Savannah has an enormous number of revisions on extremely mature and widely used OSS projects dating back decades. We obtained CVS and Subversion metadata from all projects on GNU Savannah and BerliOS that use these services. While SVN is rather new, it is being widely adopted by the OSS community and many projects have migrated from CVS to SVN, so the data must be studied along
with CVS data to get an accurate picture of a project’s development history. Additionally, java.net hosts software projects and provides CVS repositories. Apache.org is another hosting site for Apache Software Foundation projects, many of which use the Subversion repositories. If possible, we hope to obtain versioning metadata from these sites in the future.

1.3 OSS Studies

Raymond’s work is perhaps the best known of OSS literature. Chris DiBona’s Open Sources [17] (and Open Sources 2.0 [18]) is another important group of writings on OSS. Both of these publications offer a general overview of OSS and often tend to be theoretical in nature, rather than based on rigorous data analysis. Many subsequent studies have been done to confirm, modify, or argue against statements made in these publications. Conferences such as International Conference on Open Source Systems, International Workshop on Public Data about Software Development, and International Conference on Software Engineering, along with many OSS related workshops at these and other conferences are dedicated to the study of OSS.

1.4 OSS Research Collaboratories

Besides the SourceForge.net Research Data Archive [26], there are other research-oriented databases and websites with similar goals. VectorBase is a website providing tools and datasets for users to do biology research. Ensembl is another website similar to Vectorbase which is used for browsing and mining genome data from various species. While these are in different fields, there is much that can be learned from how these websites host data and allow users to access it in various ways and analyze the data with specific tools. In the OSS field, FLOSSmole [36] offers data that is screen-scraped from various OSS hosting sites. CVSAly is a tool developed
for the purpose of examining the CVS history of a particular project and providing statistical analysis of such projects [49].

1.4.1 SourceForge Research Data Archive

The SourceForge Research Data Archive is an ongoing project at Notre Dame that makes SourceForge.net data available to researchers around the world. SourceForge.net provides SRDA with monthly snapshots of their back-end database. In that database, there is information about users, projects, project development status, target audience, target operating system, programming language, download statistics, web page hits, file release information, forums, and other statistics. Actual source code is kept in Concurrent Versions System (CVS) or Subversion (SVN) repositories. This data was downloaded and loaded into a database, the details of which are discussed in chapter 3.

1.5 SourceForge.net Data in SRDA

We have an existing database (called timeline) which consists of monthly snapshots of the SourceForge back-end database. Each snapshot provides much information about every project and user at the time of the snapshot. Since users migrate to different projects and new projects are created, SourceForge’s database is always in flux. Project history, download statistics, development information, file release information, forums, and miscellaneous statistics are all contained in their database. However, actual code and coding history are kept in a separate repository. Each month, a new schema is created (following a naming convention, which includes the month and year) to hold that specific snapshot. Naturally, timeline is organized chronologically by schema [26].

The monthly snapshots are about 5GB in size, compressed (around 11GB uncompressed). The entire timeline database recently eclipsed the 600GB mark. Peri-
Periodic full database dumps are performed and a backup copy of each snapshot is also archived. The entire archive including *timeline* totals almost 1.5TB of data. Each snapshot (schema) contains approximately 80 tables, all containing between 2 and about 30 attributes. Some tables, such as users contain over one million rows.

Query access to *timeline* is made available under a license and restricted to registered users. Due to the enormity of the database and the database structure, querying can be a difficult, awkward, and time-consuming task. Initially a simple web query form was made available accepting Structured Query Language (SQL) statements. To remove an often slow intermediary of the research process (humans), programmatic access was made available via support of Web Services.

In addition, since there is no documentation of the database provided for us, we must manually investigate the data to determine what is actually held in the rows. In other words, there is much metadata that must be known before a researcher can make an intelligent query. Oftentimes the names of tables and attributes are enough, but it can be difficult to determine the relationships between tables. A wiki was deployed for the goal of automated and user-contributed documentation. Building the framework was automated, while user contributions slowly increase. Information contained in the wiki includes a list of schemas in the database, the tables in each schema, the list of schemas in which each table appears, the SQL table definition for each table, the range of values or possible distinct values in a table (when applicable), data accuracy information, and how to locate commonly studied data items.
CHAPTER 2

ADVANCES IN THE SOURCEFORGE RESEARCH DATA ARCHIVE

2.1 Introduction

The SourceForge Research Data Archive (SRDA), located at http://zerlot.cse.nd.edu, is a collection of Open Source Software (OSS) data and resources [26] [53]. Over 200 researchers worldwide use the archive for research in many fields. In this chapter, we describe the recent changes, the work in progress, and future plans for making the archive easier to use and for allowing more advanced research to be done with the data available.

We receive monthly database snapshots from SourceForge. They are about 11GB uncompressed. Each dump is a snapshot of SourceForge’s back-end database and is loaded into the timeline database as a new schema associated with that month (of the form sfMMYY). In early 2008, our PostgreSQL database eclipsed 600GB total. Project data available in the monthly dumps includes descriptive and statistical data on projects and users. Two notable missing kinds of data are mailing list and versioning (CVS/SVN) data, which will be addressed in this chapter.

We make this data available for researchers who have completed a license form. The data can be accessed through use of a web form as well as through a web service interface. A documentation and instructional wiki is provided as well as a schema.

\footnote{Some material in this chapter has been previously published in [58]}
browser. Query history and autocompletion are two features that help users make correct queries to retrieve the data they desire.

2.2 Other OSS Research Websites

In addition to the SourceForge Research Data Archive, there are other websites hosting OSS data or resources for performing research. FLOSSMole (ossmole.sf.net) [36], Qualipso [46], Qualoss [47], LibreSoft [39], related projects FLOSSMetrics [14], the CVSAnalY tool [49], and SQO-OSS [15] aim to help analysis and benchmarking of OSS projects. We have the benefit of getting the data directly from SourceForge, saving us much time and energy screen-scraping the data or acquiring it by other means. The FLOSSMole project releases data sets every month and they gather the data from multiple sources. They have the advantage of obtaining the data directly from the SourceForge pages that your browser sees. It can be difficult to find data in timeline that is easy to locate directly on a project’s SourceForge page. Also, occasionally data in timeline appears to be incongruous with project page data. FLOSSMole also has data from other sites besides SourceForge. Many of our users also use FLOSSMole to supplement their research. There are occasionally differences between FLOSSMole data and SRDA data, for numerous reasons. While an examination of such differences is warranted, that is outside the scope of this thesis. The CVSAnalY tool helps a researcher manually inspect CVS metadata from OSS projects. An examination of the SRDA along with other research repositories can be found in [40]. Bart Massey made his CVS datasets used in [41] available in arff format (Attribute-Relation File Format, a format used by the Weka program) at the Promise Software Engineering Repository [51].
2.3 Data Access

Data access is restricted to registered users. A query form (found in figure 1) is a guided form that helps a researcher form the appropriate query to obtain the data they need. The most recent addition to this page is autocompletion of text fields. Every potential valid string for the associated SQL field (select, from, and where) is listed that begins with the text already entered in the field by the user. This feature is shown in the aforementioned figure. Data format options are a few record separators as well as XML format (all illegal characters get encoded), and the option to include the query itself as part of the result set.

Figure 2.1. The query form page. The autocompletion feature is displayed.

Additionally, a SOAP (formerly Simple Object Access Protocol) web service interface is available. The SOAP request must authenticate over SSL so this is
also restricted to registered users. This programmatic access is more versatile for researchers and more efficient. Users can script as many requests as they desire and obtain results from thousands of queries in the same amount of time it would have taken them to hand craft and submit a few dozen queries using the web form.

Upon successful query completion, that query is saved to that user’s query history and automatically loaded into the fields the next time the query page is accessed. The most recent queries are displayed on the query form page in a possibly abbreviated form (they are truncated after a certain length to keep the page from being stretched) and all queries are available in their entirety on the query history page, which is found in figure 2.

Figure 2.2. The query history page. Queries are color-coded by section of the SQL statement.
2.4 Usage Statistics

Figure 2.4 shows query and download statistics for the SRDA over the past 3 years. Queries and downloads (MB) are not necessarily related since there are users who make thousands of queries with very small result sets and there are others that make queries that return multi-megabyte result sets.

2.5 Documentation

The Monthly data dumps come to us undocumented except for what the attribute and table names may indicate. To facilitate querying the data, a wiki (using the mediawiki engine) is provided which hosts researcher and administrator-contributed content. Foreign key constraints are not present in the database, but
are implied between certain tables (for example the group_id row appears in many tables, which is an unique project identifier). Every table has its own entry in the wiki along with the database table description from the most recent schema in which that table appears. There are around 80 tables that appear in every schema. Additionally, a brief description of the data contained in that table and information about data accuracy may be available depending on the table.

Other sections of the wiki include an FAQ, a page on how to find data, and a page of entity-relation diagrams created by Scott Christley.

Another documentation resource provided is the schema browser. Descriptive output is produced dynamically by querying the database with either a schema name (returns the list of tables in that schema) or a schema name and a table name (returns the PostgreSQL table description). Users can query from the top level links to each schema or use the query field to enter a query of one of the following formats: *schema.table*, *schema*, *table*. If just a table name is entered, that table's
description will be retrieved from the most recent schema. This capability is placed on the right side of the query page to ease table lookup. The schema browser can be seen in figure 5.

2.6 Additional Data

In addition to the descriptive and quantitative statistics available from the SourceForge data dumps, there are two often requested kinds of data missing, specifically CVS/SVN versioning data and mailing list data. In chapter 3, we describe the acquisition of CVS (Concurrent Versions System) and Subversion metadata from all SourceForge projects that use those resources and allow public access to them. That data has been loaded alongside the SourceForge data dumps. This metadata is important to OSS researchers. Versioning metadata consists of who made a change (a commit), when the commit was made, the files that were changed, a user-supplied comment, and in the case of CVS, the number of line changes made to the file [9]. Subversion is relatively new, but its adoption is widespread throughout the OSS community and many projects migrated from CVS to SVN (potentially creating difficulty for those studying such projects). This data is now available for querying along with the SourceForge data dumps. More details of the CVS/SVN metadata and the database it is stored in can be found in chapter 3. Entity-relation diagrams for the versioning metadata are available in figures 3 and 4.

2.6.1 CVS and SVN Research

Versioning metadata is an often-used and valuable OSS project resource. In [28], CVS metadata is visually depicted from both a file and author perspective. In both [27] and [33] the authors graph data chronologically and examine the number of modification requests (MRs) in a particular timespan. A modification request is equivalent to a CVS or SVN code commit (check-in). Grouping file changes into
one atomic commit is important because it represents one user committing changes to possibly more than one file. This can be useful for determining patterns, such as which files are closely related. While SVN logs are inherently grouped in this manner, CVS logs are not and require careful inspection to determine which files, if any, were committed simultaneously. In [33], they also display social network statistics (which can be determined from the CVS or SVN users data) at different points in time. In [5], the researchers use a tool called cvs2cl to group CVS logs into change logs, which can allow a better grasp of code changes [21]. Number of CVS commits has been used as a metric for project activity [34] [35]. In [12], CVS activity and many other metrics were used to cluster contributors. In [16], CVS data was studied in various ways in conjunction with data from the SRDA.
2.6.2 User and Project Data and Connections

CVS and SVN user-project networks can be created for any point in the history of the data. Since it may be useful for a researcher to look at how this network changes over time, it is essential we provide this capability and it can be calculated fairly easily. In the pre-processing phase, first and last commit time are recorded for each user for each project. This data is stored in the user_group tables. With this data and a supplied time, we can determine the user-project network at that time with minimal overhead for small networks or cliques by checking if the supplied time is within the first and last commit window. This serves another purpose in that we can easily compare versioning data to timeline data which is tied to a particular moment in time.
2.6.3 Other Open Source Hosting Sites

While SourceForge is the largest OSS development website, BerliOS Developer and GNU Savannah are some other popular OSS hosting websites. BerliOS is a German website with about 1500 users and nearly 3200 projects. GNU Savannah is another hosting site that started when the SourceForge project itself was relicensed as proprietary software. While SourceForge has over 160,000 projects, these small and potentially tight-knit hosting sites offer another rich set of research data. GNU Savannah has an enormous number of revisions on extremely mature and widely used OSS projects dating back decades. We obtained CVS and SVN metadata from all projects on GNU Savannah and BerliOS that use these services. Additionally, java.net hosts software projects and provides CVS repositories. Apache.org is another hosting site for Apache Software Foundation projects, many of which use the Subversion repositories. If possible, we hope to obtain versioning metadata from these sites in the future.

2.7 Administrative Concerns

For each month when we receive a new data dump, there are hundreds of wiki pages that need to be updated. The reason for regularly updating wiki pages with static data is to show which schemas each table appears in and to display the most recent instance of each table since occasionally minor changes are made to them by SourceForge. Most of these pages need to be updated in a similar way in similar places. To ease this process and improve robustness, updates to many pages are automated. Using a mediawiki versioning tool called mvs (http://search.cpan.org/˜markj/WWW-Mediawiki-Client/bin/mvs), which provides a command line client, all necessary wiki pages are retrieved and updated using perl scripts. Updates made to other parts of the wiki pages by users remain intact after this update process.
The registration process consists of a scholarly researcher completing a form (which can be found through the wiki for those interested in access requirements and data restrictions) and faxing it to the Principal Investigator (Greg Madey). Then a user is created through the wiki and finally, results directories and symbolic links are created. An authentication module is used by the server to authenticate users through the wiki database when accessing the query page. This has the added benefit of using the wiki infrastructure for password management. Obviously, the same username and password are used when making changes to the wiki. A user is manually added through the wiki and the rest of the process is automated through a MySQL trigger and a perl script.

2.8 Features in Development

A development server was recently deployed, which should help speed development of new and current features. In this environment, major changes can be tested without worry of interrupting researchers who may be scripting thousands of queries or investigating table information. Features in some phase of development are detailed in this section along with the benefit they will provide.

2.8.1 Stored Queries

Although the wiki provides a Web 2.0 aspect, we can take things further. Users must query the database using SQL, although many of our users are newcomers to the declarative language. Potentially, there is overlap between the queries researchers are performing. Stored queries would allow users to access their own or other user’s previously executed queries that they thought others may find helpful. For example, here is the query to obtain operating system-related information on project with group id 235:
This query is difficult to remember and may be difficult to distinguish from similar looking queries that retrieve programming language data, for example. When the stored queries feature is complete, a user will be able to store a query, give it a descriptive name, and identify the variable part (in this query, it is the group_id 235). Then a researcher can access a list of these queries, load one, enter the variable data specific to the project or user they wish to retrieve information from, and execute the query. By default, queries will be public, with the option to hide your queries from other users. Additionally, users will be able to comment on the reliability of a query and view the popularity of a query. This feature will lower the learning curve for new researchers and encourage others not to repeat work that has already been done (typing in a complex query).

2.8.2 Automated Graph Production

In [27], numerous graphs are provided which visually depict the development history or current state of an OSS project. Researchers will likely be graphing data obtained from the database. Automating this process will save them time and allow them to see interesting trends earlier on in the research process. Versioning metadata plotted over time is one such interesting category that is easily automated given our database structure. Sample graphs are shown in figures 6 and 7 where cumulative line changes over time are graphed for the gcc compiler and the emacs editor. Cumulative changes are shown instead of number of changes because when plotted over a large range of time, it is difficult to see smaller changes. A cumulative graph shows sustained development more clearly. Both the gcc and emacs projects span many decades of development.
Figure 2.7. Cumulative line changes over time for the GNU C Compiler. The first tick is Nov 1985 and each tick is approximately 3 years and 2 months. 1.2e9 is Jan 2008.
Figure 2.8. Cumulative line changes over time for the emacs editor. The first tick is Sep 1982 and each tick is approximately 3 years and 2 months.
2.8.3 Web Service Improvement

The SOAP interface is useful from a programmatic standpoint, but currently limited. The web service actually just executes a query. The user must retrieve the results (which can also be scripted). We do not provide a WSDL file for users, but rather a wiki page with some sample client code they can use. Result format is also restricted to text with a user-specified field separator. Result sets in XML and programmatic schema browser access are being developed to provide a more complete, cleaner, and more robust web service interface to the database and the new versioning metadata tables.

2.9 Planned Features

Result sets can be very large, often around 20MB of text. For such sets, local examination (and probably postprocessing) is necessary. Attempting to view such sets can hang a browser. Result pagination or smart inspection is a planned feature to ameliorate this shortcoming. Such a feature would also allow for visual postprocessing and reordering. Another shortcoming of the archive is that results simply clobber the most recent results. That is, all queries output to the same two files (one for text results, one for XML results). This was a naive default decision to deal with space constraints in an easy to implement manner. However, this means queries are potentially being repeated if a user forgets to download results before executing the next query. A helpful feature would be to store results in a database and allow users to name result sets or provide short descriptions for later retrieval. This would help researchers from a logistical standpoint.

Another improvement to be made is to the autocompletion feature. Currently, it is naive and will offer completions that make sense syntactically but not logically with regards to the database (e.g. select a field name from a table that does not have
that particular field). A smarter solution is to parse text entries asynchronously and only return relevant and valid options for the autocompletion feature. Additionally, we are in the process of obtaining mailing list data. Conceivably, the only other widespread data related to SourceForge to be obtained after this is source code itself, which is another long-term goal of the SRDA.

2.10 Conclusions

In this chapter, we described the state of the SourceForge Research Data Archive, a valuable resource for Open Source Software research. Notably, new versioning metadata for over 100,000 projects has been added and features were implemented to ease the research process for our users. Features and resources under development were described as well as plans for the future. Current work should yield helpful user contributions in the form of stored queries.
CHAPTER 3

WAREHOUSING, MINING, AND QUERYING OPEN SOURCE VERSIONING METADATA

3.1 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 chapter, we describe the downloading and warehousing of such data and the interface and resources we offer for studying the data. We also present some preliminary data analysis. The process is similar to that done in [20] 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 [70]. Xu built a similar retrieval framework however for web pages to gather project statistics.

All the work outlined in this chapter \(^1\) for the OSS Versioning Metadata Database is towards the goal of enabling and supporting research in areas such as software engineering, open source phenomena, social network analysis, data mining, and project management. Since many of the 200+ researchers worldwide using the database do not have a background in computer science, querying must be simplified so that SQL syntax knowledge is not required. In addition, useful research tools and infor-

---

\(^1\)Some material from this chapter appears in a working paper [59].
ination should be made available to the researchers. Graphical, statistical, and data mining tools with easy to use interfaces will greatly facilitate the research process. For more technically knowledgeable users, fine-grained control of queries and data mining software parameters must be made available in addition to the ability to script or automate these tasks. The newly-formed database containing CVS and SVN metadata offers new research opportunities for large-scale OSS development analysis. An associated website will link to the database and offer research resources. Since we are using this site for further research on open source software, we are both developers and users. Any feature that we find useful for our own research will be added for others to use in the future.

3.2 SourceForge.net Data

Most of SourceForge’s data is stored in the back-end database, archived in timeline. The actual source code is stored in a Concurrent Versioning 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 metadata (everything except the actual code) and built another database that will be juxtaposed with the timeline database. The SVN metadata was also obtained from SourceForge. The log files containing this metadata are fundamentally different between CVS and SVN, and as a result will need different database designs.
3.2.1 Existing Research on CVS Data

In [27], they analyze numerous statistics related to modification requests (MRs). First they plot the number of MRs per month over the lifespan of the project. They also plot the cumulative percentage of transactions by developers, ordering the data from most active to least active. They plot similar data for number of files per MR. Lastly, they analyze the most often edited files, the top developers by activity, and activity categorized by file extension. Automatically producing these and similar graphs will be integrated into our site. In [33], they apply social network analysis to CVS data. They analyze the committer network as well as the module network (modules being a subset of a project) by computing degrees, weighted degrees, distance centrality, betweenness centrality, and clustering coefficient (weighted and unweighted). A similar approach is taken in [37] to group contributors into core and peripheral teams. Since social network analysis is a field that often studies software development, such as in [12], [69], [68], [11], [25], [13], and [57], focus on this area is essential towards making a useful site for researchers. In [23], Gao studied the social network of SourceForge and mentioned in future work that a similar study with GNU Savannah would provide more comprehensive conclusions about the OSS social network. We now have the data to conduct such an analysis. Another shortcoming of that thesis was incomplete network data of the SourceForge community. The versioning metadata we obtained provides a complete history of the SourceForge social network as it relates to code committers. Gao also presented results of network research on the SourceForge community in [24]. In [2], the researchers mine CVS and SVN data for bug-introducing changes. Mockus and others performed in-depth case studies of Mozilla and Apache using CVS data among other information [42]. Scacchi examines development practices in the OSS gaming community and how CVS plays a role in project management [52]. Canfora and others examine the
CVS/SVN logs in conjunction with source code to follow OSS project evolution [8]. Voinea and Telea visually mine CVS repositories and use a CVS data acquisition tool in [65] [64]. In [10], Chen and others align CVS log data with source code for annotation and browsing purposes in a tool called CVSSearch. In [72], CVS log data were used to identify logical couplings of files. A similar study was done in [71] by Ying, et al for the purpose of identifying relevant related files that may need changing when a MR is made. Van Rysselberghe studied CVS log data to mine what they phrased “frequently applied changes” in [61]. He also visualized CVS log data over time in [62]. Mockus studied version control logs to classify software changes [43]. Atkins used version control data to analyse how software tools affect development [1]. Graves and Mockus described a methodology for inferring code change effort from version control information [32]. Eick and others examined “code decay” of a 15 year old piece of software by analysing change information in version control logs in [19]. Gall and others studied CVS log data to identify dependencies in [22]. In [31], the researchers used information from change management systems to predict when faults will be introduced into code.

Some areas in which further research can be done include temporal analysis, such as bursty behavior of commits, social position changes (as measured by type of activity being done), and social network evolution. With the addition of general economic information, we may be able to identify economic changes with trends in OSS activity. Commits can be classified through the use of log information, which has been done to some extent already, but often along with actual source code. Separate OSS networks, such as SourceForge and Savannah, can be compared in numerous ways to identify any fundamental differences. Further details on possible future work can be found in chapter 4.
3.3 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 [55]. 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) [9].

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
```
3.4 Download Process

In order to achieve timely CVS log downloads, minimize the load on SourceForge’s CVS servers, and not appear to be a denial-of-service attack, the following steps were taken. Initial inspection showed that SourceForge employs about a dozen CVS servers. This has an obvious effect of distributing CVS transactions and minimizing load on any specific server, which is undoubtedly the reason SourceForge set up their hardware thusly. An obvious solution is to query the projects serially thereby keeping the maximum stress on any one server to one transaction at any given time and since different projects are distributed apparently at random over their set of servers, consecutive requests to any given server are unlikely. This would certainly minimize strain on their servers. However, rough calculations showed this would take a few weeks.

To alleviate that problem, many requests can be made simultaneously from one machine (using threads or processes for each serial line of requests). This would certainly go much faster since the rate-determining step occurs on the CVS server. However, this results in one machine possibly making multiple simultaneous requests to the same machine. Depending on the amount and frequency of requests, this has the potential to be erroneously flagged as a denial-of-service attack.

The solution 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.
3.4.1 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. 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 was deployed to monitor the database. Since the database was being constantly updated, a static web page with user-initiated page refreshes would be tedious and impractical. Automatic refreshes are unsightly and unnecessary. An AJAX webpage updated automatically at a fixed interval to check for updates and then update the relevant portions of the page in a smooth manner. Running tallies and recent entries for each category (in progress, successfully completed, initiation failure, and progress failure) were displayed. A schematic of this process is shown in figure 3.1.

In order to ensure smooth interaction between the handler processes and the central database, synchronization measures were taken. One database table was used to hold all of the information. A handler process would take an unassigned project, then update its status to in progress and set the server name to the appropriate VM, and give it a timestamp. Upon return, it would update it again to set its new status, add another timestamp, and if appropriate, record the number of lines in the log file that was just retrieved. A semaphore file was employed allowing only one process to have control of the file at a time. Once control was gained, the
process could update the database at will and no other updates could be made to
the database by another process during that time. Once the appropriate database
updates were made by a particular process, it would release the semaphore file and
another process could pick it up.

Each VM was loaded with SOAP (a Web Service protocol) server code (using
the Perl SOAP::Lite module) which would receive a CVS project name and a flag
indicating if it should stall or not. Since SOAP requests are naturally made from
a client to a server, it was natural to have the central server making client requests
to remote servers. This way job submission could be controlled entirely through
the central server. Each handler process would submit a SOAP client request to
its assigned VM. The stall flag was also assigned on the central server side since
the VM SOAP server code was stateless, which greatly simplifies the code on the
VM side, and the handler process on the central server was naturally stateful. The
handler process knew the SourceForge CVS server that was just queried and it can
easily check if the next project is also hosted on the same server.

To make the SOAP server code intelligent, the SOAP server code would fork with one process initiating the CVS log download and the other monitoring the download progress. At fixed intervals, the monitoring portion would check the progress of the download via a line count. If the line count has not changed after a reasonable amount of time, the process was assumed to be hanging. It would then be killed and a corresponding message would be returned to the central server. The handler process would record a progress failure in the database for that project and then submit a new SOAP request. This safeguard proved to be very important towards completing the download process in a timely manner. Similarly to the progress failure, occasionally a CVS log download process could not even start. This may happen for a few reasons, including disabled anonymous CVS access, temporary network failure, or other server problems. Whenever the problem was not the first of those, CVS log file download would usually be successful upon a retry. An initiation failure would be detected by the SOAP server code by counting lines after the download process had completed. If the file had zero lines, it was assumed to be an initiation failure since even a blank CVS template had lines (as it turned out, the same number of lines for all projects that had no contributed CVS commits).

For fine-grained control over the download process, a method was implemented allowing for graceful shutdown. A file containing a 1 or a 0 indicated whether or not the central server should continue to submit SOAP requests to the virtual machines. This flag was checked in each handler thread before submitting a request. If it was set to 0, it would gracefully exit and not submit a new request. We wrote a basic toggle program which would flip the value from 1 to 0 or vice versa. If we noticed a problem, we could run this program and each handler thread would exit after receiving the return information from the appropriate virtual machine. This graceful
TABLE 3.1

NUMBER OF LOGS DOWNLOADED FROM EACH HOSTING SITE, CLASSIFIED BY VERSIONING SOFTWARE

<table>
<thead>
<tr>
<th>Hosting Site</th>
<th>CVS</th>
<th>SVN</th>
</tr>
</thead>
<tbody>
<tr>
<td>SourceForge</td>
<td>103869</td>
<td>24416</td>
</tr>
<tr>
<td>BerliOS</td>
<td>1252</td>
<td>1718</td>
</tr>
<tr>
<td>Savannah</td>
<td>1775</td>
<td>8</td>
</tr>
</tbody>
</table>

shutdown kept the database from being in a state that did not accurately reflect the progress of downloads. Killing the handler threads while a request was operating would leave the database entry for that project in the “in progress” state, but the handler thread would not be able to set it to a different state upon completion.

We used the toggle program so data would only be downloaded at certain times. We started using one virtual machine, then slowly added the others. We also primarily ran the download program at night at first. When there seemed to be no problems, we let it run for most of the day, shutting it down when it seemed to be going particularly slow (assumingly when SourceForge had the most traffic). 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.

3.4.2 BerliOS and Savannah Downloads

BerliOS and Savannah data was downloaded in the same way although with minor changes. A larger stall time was implemented (15 seconds) and since all requests went to the same remote IP address, the VM always stalled before making the next request. Since these sites did not have multiple servers with different IP
addresses, we tried to leave a smaller footprint. Since the amount of data downloaded was significantly smaller than that from SourceForge, obtaining data for all relevant projects still went very quickly.

3.4.3 Continuous Future Updates

The CVS and SVN data is already many weeks out of date. 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 [50]. 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. How often log data updates will be run has yet to be determined. Although far less data will need to be downloaded, it is still putting strain on the server. Further testing needs to be done.

3.5 Parsing the Logs

The format of each file is plain text and as a result, delimiters can appear in the context of the log as something other than a delimiter. As a result, simple assumptions could not be made about the format of a log file. For example, a string of a precise number of dashes followed by a new line is the delimiter for separating revisions in CVS logs. However, the same string can (and often does) appear in the commit message section of a log file. As a result, regular expression features such as look ahead and look behind were used.
3.5.1 CVS

The CVS log file format is as follows:

```
RCS file: /path/to/filename,v
head: 1.2
branch:
locks: strict
access list:
symbolic names:
    meaningful_name: 1.2
    another_name: 1.2
    yet_another: 1.1
keyword substitution: kv
total revisions: 2; selected revisions: 2
description:
```

```
revision 1.2
date: 2005/10/15 17:35:14; author: username; state: Exp; lines: +4 -4
User entered commit message for revision 1.2
```

```
revision 1.1
date: 2004/10/15 17:35:14; author: username; state: Exp; lines:
User entered commit message for revision 1.1
```

The RCS file line is a unique full path to the RCS file (.v extension) in question. When checked out, the file will not have the ".v extension." The head is the most recent revision in the repository. “Total revisions” is the number of all revisions in the history of the project for that particular file. “Selected revisions” is the same except when log files are filtered for certain info. Since we obtained logs in their entirety, total revisions and selected revisions were always equal. The description field is an optional field with an overall description of what the file is or does. The other field we were interested in from the header portion was symbolic names. Each line in the symbolic name field is a user-provided tag (or release name) followed by a revision number. This says that a given tag is associated with a specific revision of this file. For example, a developer may work on a number of different files, and
the most recent revision numbers may all differ. In other words, one file may have 2 revisions and another may have 25. Assuming no branches, the former will have a revision number of 1.2 and the latter 1.25. To group these revisions together, a tag is applied. As shown in the example above, a particular revision may have more than one symbolic name associated with it. A rarely edited file in a project (perhaps a README or INSTALL file) may have dozens or hundreds of symbolic names associated with the same revision of that file, while others (the core software files) may never have more than one symbolic name for a particular revision.

After the header portion of a CVS log entry, specifying the filename and symbolic name, specific revision history is provided. Starting with the revision name, then followed by a timestamp, the user who committed the change, the state of the file, the number of lines added and removed (if applicable) and the commit message associated with that revision. For a newly added file (known as an initial commit), number of line changes are not listed since there is no previous version to which to compare. Another optional portion is the branches section of a revision. If a revision is branched, the branch line(s) will be listed immediately below the revision number line. Revisions are of the form branch.revision where revision starts at 1 and increases monotonically. A revision has an even number of dot separated numbers, while a branch has an odd number of dot separated numbers. A branch is started off of an existing revision. If the revision branched off of is 1.2, the branch will by default be 1.2.1 and from there, the revisions will be numbered starting with 1 (1.2.1.1) and increase to 1.2.1.2, etc. Those revisions may also have branches off of them. Symbolic names may point to branched revisions. Below the row of equal signs, a blank line will follow and then another file entry will begin with the RCS line.

In an examination of over 100,000 CVS log files, both the revision separator
(series of specific number of dashes) and file separator (series of specific number of equal signs) themselves appeared in the commit message or description portion of the log file. This made distinguishing between a commit message portion and an actual separator very difficult. The parser started out simple and clean, then had to be revised to account for the ugly situations where separators appeared in commit messages.

Lastly, it must be mentioned that CVS allows a user to check in multiple edited files simultaneously. Each will appear as a new revision for that particular file and the same commit message will appear in the logs for each file. CVS log files are not organized chronologically and it is not obvious what groups of files have been checked in simultaneously without careful inspection of commit timestamps, a problem encountered by Massey in [41] in his analysis of X Window System and Nickle. A solution to this problem of determining which commits are part of an atomic commit is used in [74]. Zimmermann and Weigerber in their analysis use a fixed time window and sliding time window with a threshold of 200 seconds. Both of these options can be easily accommodated for through the web interface to the new database.

3.5.2 SVN

The Subversion (SVN) log file format is as follows (svn log --verbose):

```
r26 | username | 2005-11-16 18:02:43 -0500 (Fri, 16 Nov 2007) | 1 line
Changed paths:
   M /trunk/backend/imap.php
   A /trunk/much_better_name_fig.eps (from /trunk/poorly_named_fig.eps:2)
   D /trunk/no_longer_needed.txt
Comment for revision 26
```

```
r25 | username | 2005-11-16 16:51:41 -0500 (Fri, 16 Nov 2007) | 1 line
Changed paths:
```

35
SVN improved upon CVS, removing many of its drawbacks (some drawbacks are outlined in a paper by van der Hoek which predates the initial release of Subversion [60]). Fundamentally, the log files are now organized chronologically. They also remove any need for multiple symbolic names for any given revision of a file. In CVS, when a file was edited and committed, only that file was updated to a new revision number. With SVN, every update, regardless of the number of files committed, causes an update in the revision number. This revision number refers to the current state of all files. In the framework of CVS, it would be like making a monotonically increasing number tag after every commit. Every file changed on a commit will appear in the log when the --verbose option is present. The letter M preceding a filename indicates that file was modified since the last revision. D means the file was deleted and A means a file was added (or renamed). If it is renamed, it will indicate the previous name of the file following its new name and path. The lines portion indicates the number of lines in the commit message, which follows the first blank line after the changed paths portion. As a result, the SVN logs were far easier to parse correctly and reliably than the CVS logs. Their chronological nature also means less post-processing is necessary (unless viewing individual file history is needed, although that is still a relatively straightforward computation). Revision separator is a specific number of dashes, similarly to how CVS separates revisions.
3.6 Designing the CVS Database

Traditional database design issues were considered here. Tradeoffs between space and query speed were also considered. Redundancy across tables makes multiple copies of data but may substantially speed up a query that would otherwise have to perform a lookup in another table. All data was placed into a PostgreSQL database. All text fields are of the PostgreSQL type `text`. A set of tables was created for each repository – SourceForge.net, BerliOS, and Savannah – for both CVS and SVN.

3.6.1 Main Table

Each file has one or more revisions. Each revision of a file has zero or more symbolic names. Each file has exactly one header portion. Logically, one table is needed to store the header information for a given RCS file (one entry for each). The RCS filename (with full pathname) is unique. As such, it is a primary key in our database. In addition to the information in the header, the unix group name (project name) is added to each entry here to group the entries together logically. Some of the information has little research value and is not included in our database. The fields included and their types are listed:

```plaintext
file_name text
head text
total_revisions integer
branch text
description text
unix_group_name text
```

3.6.2 Revisions Table

Each revision refers to exactly one RCS file. As such, it is a foreign key constraint in our table (referencing the main table). All other information is stored in our database.

```plaintext
file_name text
```
The timestamp is converted from the date format presented in the example CVS log file into standard Unix time (seconds since the Unix Epoch). This makes sorting by time and comparing timestamps of two revisions trivially easy, since a simple numeric comparison is all that is necessary. Here there is no single unique field, but the pair of file name and revision is unique. As such, it is a primary key constraint in our database.

3.6.3 Symbolic Names Table

The symbolic names (tags) table contains a file name which references the main table, and therefore is a foreign key constraint. The other two fields in the table are revision and symbolic name. This pair is unique over all columns and is a primary key constraint in our database.

3.7 Designing the SVN Database

The same issues were considered when designing the tables that would hold the SVN data. Due to the reorganization of the infrastructure when compared to CVS, fewer tables are needed. Arguably, the most interesting research information is presented more clearly and logically.
3.7.1 Revisions Table

The revision number is unique for a given project and is a primary key. Other information stored is the username, timestamp, size of commit message, and the commit message itself.

```plaintext
unix_group_name text
revision integer
user_name text
time integer
lines integer
log_message text
```

---
r26 | username | 2005-11-16 18:02:43 -0500 (Fri, 16 Nov 2007) | 1 line
Changed paths:
  M /trunk/backend/imap.php
  A /trunk/much_better_name.fig.eps (from /trunk/poorly_named.fig.eps:2)
  D /trunk/no_longer_needed.txt

Comment for revision 26

3.7.2 Files Table

Since more than one file may be committed for a given revision, another table is necessary to store them without unnecessary data duplication. The revision number references the main SVN revisions table, and therefore is a foreign key constraint. The type of modification (e.g. M, D, or A) is stored for each entry as a single character. The appropriate file name is stored as well. If the file was moved from somewhere else, the original file name will be stored as well as the revision number that it came from. For an entry like a simple modify, these fields will be null.

```plaintext
revision integer
type character
file text
original_file text
original_rev text
```
3.8 Creation of New Tables

After considering important database design issues and laying the plans for the above tables, new tables were formed from the existing data. While most of the information is in the database, querying it may be unnecessarily long and in some cases it may be infeasible.

3.8.1 User and Project Data

A researcher often wants to obtain a simple list of all projects in the database. To obtain this information, we can select the distinct unix_group_names from the main and revisions table for CVS and SVN respectively. However, this can be a time-consuming query that may bind up the database server unnecessarily. It also may be of interest to compare the list of all group names with another table. It makes sense to have this information stored in a table of its own. It is logical to store the number of developers on a project in this table. Similarly, there should be a users table. At this point, another table can be made from these two new tables, specifically a user_group table indicating the connections between users and groups. In this table, a row exists in the database if that user works on that particular project. In addition to a simple list of groups, a researcher may start off by obtaining the list of all contributors for a certain project. This same table can also easily provide a list of all projects on which a specific user has worked. While the users, groups, and user_group tables are available in the monthly snapshots we obtain from SourceForge, this information must be created for BerliOS and Savannah. For the users table, it is convenient to store the number of (file) commits that user has made.
3.8.2 Time Series Organization of CVS Data

For CVS data, it is often of interest to examine the data chronologically. The researchers [27] and [33] graph data chronologically and examine the number of modification requests (MRs) over a period of time. Some of the things they plot here are number of MRs per month. In [33], they plot numerous social network statistics at different points in time. To ease the production of these graphs, new tables were created. A table holding modification request timestamps, username, group name, and number of files was created. This allows for easy and efficient extraction of MR data. For social network statistics that are time-related, a table was created where each row contains a username, a project name, the timestamp of their first MR for that project, and the timestamp for their most recent MR for that project. This timestamp range represents a window where it can be assumed that user is working on this project. From this data, it is easy to obtain a snapshot of who was working on a given project at any given time. Additionally, it may be interesting to researchers to see how the file tree evolves over time for a project. A table containing filenames, along with the timestamp of when that file first appeared has also been added. Statistics such as total number of files over time can easily be obtained from such a table.

3.9 Querying the Data

A complaint we often hear from researchers about the SourceForge.net Research Data Archive is that it is difficult to extract the data they need. A SQL command form provided where a user must form a valid SQL query and results are presented either as text or XML. Table metadata is provided, but it is mostly a table description that PostgreSQL provides. Interpreting the data can be a difficult task. Such a query form can be classified as bottom-up. In other words, a researcher may have
to query multiple seemingly unrelated tables to get something like the development status of a particular project. A far more intuitive approach which is easier to use from the researcher’s standpoint is a top-down approach. A user should be able to ask for the development status and the logic of the form should deal with the messy details. In this way, the researcher interface should be an easy to use, yet powerful layer of abstraction built on top of the database. The CVS and SVN data will eventually be juxtaposed with the SourceForge.net Research Data Archive for the purpose of performing more powerful and interesting queries, but for now they stand separate. Learning from the drawbacks of the Archive, a new approach was taken. Researchers can search by user or project name and from there, more detailed information is provided. When the search term is a user, a list of projects that user has worked on is returned. When a group is searched for, the users who have contributed are listed, then a list of distinct commit times (MRs) are provided along with cumulative line changes, which is graphed immediately below that. The number of total commits and lifetime of the project are provided. Also, the number of MRs are binned by month and graphed. Also, a list of the all files in the project is provided. To make it an intuitive interface, nearly every piece of information returned is a link to another page. Each link goes to what the user most likely suspects it does. For example, clicking on a file name will list all revisions of that file in a format similar to the CVS log file. On that page, username and time are links. Clicking on a username will show all commits by that user for this project. Clicking on a timestamp shows all commits that occurred simultaneously at that time. Each page has upper level links, bringing the user back to a more general view of the project information. As a result, it is a powerful and intuitive CVS history browser with automated graphing of certain statistics.
3.9.1 Planned Additions

Since this is an evolving project, numerous additions are planned to make the site more useful, efficient, and easier for researchers. Data mining and social network analysis tools will be integrated to provide more features and tools that researchers commonly use. Fine-grained control over graphs will allow a researcher to create a graph exactly of the form they want. Social network graphing tools will be integrated so subgraphs of the developer, project, or bipartite developer-project graphs can be viewed or downloaded for inclusion in research papers. Filters over username, project, time, or a certain number of files will give the user a more manageable view for visual inspection or studying statistics over a specific cross-section of a project. The possibilities in that area are numerous. Also, aggregate data over all of BerliOS, Savannah, or SourceForge projects (and how the three hosting sites differ) will be of great interest to social network researchers and software development analyzers who want cumulative statistics.

3.9.2 SQL Capability and Stored Queries

Since all possible queries cannot be provided by an intuitive layer of abstraction, SQL access will still be provided. To make this aspect more usable, query storage capabilities will be implemented. Since a researcher using the SQL command line will likely be performing highly specialized, detailed, and often lengthy queries, this feature will allow the researcher to reuse and alter existing queries and store a brief description of what that query does. Since community contributions are often the most important part of a website (for example, Wikipedia), allowing other researchers to view all stored queries will allow them to avoid redoing difficult work should it have already been done by another researcher. As with our SourceForge.net Research Data Archive, table descriptions as well as user and ad-
ministrator contributed data descriptions will be available. If particular queries are extremely popular, they can easily be moved to the rest of the website where the query will be run under a layer of abstraction. In this way, a stored query can act as a feature request.

3.9.3 Automating Graph Production

Since many graph programs are difficult to use, this site will provide fine-grained control of many aspects of graphed data. Many typical options will be provided (such as bar graphs and line graphs) as well as customization of the range of data plotted and details like labels. This will also be integrated with the SQL command line results for more customization.

3.10 Privacy

Just like any study of publicly-available data where user-specific information is present, privacy must be considered. In [44], the issue of studying open source software ethically is raised and examined. In that paper, they examine issues of data mining and qualitative data. Since activity metrics will be produced by the database and corresponding website, any negative repercussions to any user must be considered. Many aspects of this data must examined for their effect on privacy and reputation. While this data is publicly accessible and projects can forbid anonymous CVS access, it is possible and even likely that certain users are unaware of the extent of the public nature of their work. Another factor is that usernames do not necessarily identify a particular person. However, on the other side of this fact, this username is tied to that person’s online persona. Data mining may reveal secondary information that is personally revealing even though the actual username may be anonymized. There is a similar problem in database research which shows that privacy in data is often difficult or impossible to achieve without making the data
unusable. In [63], Vinson and Singer examine the ethical issues of studying publicly available OSS data. Their paper boils down to removing identifiers (for users and groups) and, if possible, obtaining consent from users or groups.

Certain projects had names implying obscene purposes, however even a project name is very visible on a hosting website and if the creator and contributors did not want themselves associated with a potentially embarrassing project, they would likely not have used a hosting site. Another issue which is a little more difficult to analyze from a privacy standpoint is commit messages. A brief glance showed many with curse or otherwise obscene words. Committers may assume only their colleagues would see these messages and they may not want this information to be more public than it is. It is infeasible to screen such words or phrases from commit logs. However, if a commit message search capability is implemented, it would be relatively easy to remove such potentially embarrassing words or phrases from the search index. This way all log messages are available intact, but a nosy researcher cannot easily find out the developer who swears most often in log messages.

Another issue that must be considered is that many of the users are already public figures in the open source software community who have well-known usernames. For example, Richard Stallman, founder of the Free Software Foundation, appears in the GNU Savannah database with username rms. The most active contributor to Savannah projects is Ulrich Drepper, another open source figure. Roland McGrath is also a large contributor known in the OSS community. Many of the contributors to software hosted on GNU Savannah have short biographies listed on the GNU website. For them, much of the privacy regarding the open source software they have written has been minimized, often by their own volition.

Users of SRDA must agree to a sublicense before they can access the data. The terms and conditions mostly aim to protect the users of SourceForge from abuse or
embarrassment. Any data to be presented (which is itself limited to 2KB before special approval is required) must have user-related data anonymized. The data is only available for scholarly research. The terms and conditions of the sublicense can be found in Appendix 3.

3.11 Data Analysis

In this section, we will present some quantitative data on a handful of extremely mature and long-lived open source projects from GNU Savannah as well as some of the most active projects on BerliOS.

The BerliOS project cp2k is a fortran program for performing molecular simulations. It has 18 contributors over the lifespan of the project, going back to late June 2001. There have been 7912 file commits for the project. The data is found in figure 3.2.

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 3.3.

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 3.4.

The libc project, hosted on Savannah, is the GNU C library. While there are fewer contributors on this project than other mature GNU projects, the project has 94,092 commits in its 20+ year lifetime. The graph can be found in figure 3.5.
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. 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.

From the developer standpoint, we can glean different information. While the cumulative project statistics show how people contribute together, a single user often contributes to multiple projects. The same type of graphs are now plotted from the developer standpoint. The information for Savannah user rms is in figure 3.6.

The aggregate commit information for the most active user on Savannah, drep-per, is in figure 3.7.

We can also look at the activity of a user on a particular project. Here is the activity for user rms on the gcc project.

It will be interesting to see if user contributions tend to cluster in a large group and then almost completely stop for a particular project, how many projects a user works on simultaneously, and if the projects a user works on throughout their coding lifetime have any interesting connections. Also, armed with the number
of line changes (both added and removed), can we classify commits into certain
categories? Can we identify a bug fix or a patch and differentiate them from core
software additions?

3.12 Conclusions

This large data set offers a multitude of open source software and social net-
working research opportunities. We can learn about project development trends and
group similar projects together by development similarity. We can examine contri-
bution 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.
Figure 3.2. Berlios project cp2k
Figure 3.3. Savannah project gcc
Figure 3.4. Savannah project emacs
Figure 3.5. Savannah project libc
Figure 3.6. Savannah user rms
Figure 3.7. Savannah user drepper
Figure 3.8. Savannah user rms on project gcc
CHAPTER 4

ANALYSIS AND CONCLUSIONS

The SourceForge Research Data Archive is a valuable resource for OSS researchers worldwide. The additions made over the past few years have been outlined along with work in progress that will allow new and important research to be done towards understanding how OSS is developed. CVS and Subversion metadata from SourceForge, Savannah, and BerliOS has been downloaded using a robust distributed download management mechanism and stored in a database juxtaposed with the timeline database. This makes SCM research of such breadth available for the first time.

4.1 Future Work

In addition to the features currently being implemented or planned for the SRDA, much new research can be done with the new CVS and Subversion data.

There is a plethora of statistical analysis, social network analysis, and evolutionary studies that can be done.

4.1.1 Bursty Behavior

Do CVS and Subversion commits display bursty behavior? Certain time-dependant data follows bursty behavior, such as in [38]. We can study versioning data over specific timeframes to determine if it exhibits bursty behavior. This would tell us
something about the nature of OSS contributors. Kleinberg notes in [38] that e-mail exhibits bursty behavior. Since many OSS project contributors communicate via email, it is possible that code commits may also follow this behavior.

4.1.2 Social Positions Over Time

In [12], social positions were examined over a time period of a few months starting with the beginning of a project. These positions evolved in an interesting manner over just a few months. This study was done from a project-centric standpoint. A similar study over a user’s lifetime may display interesting behavior from a user-centric standpoint. Do users hold the same social position over their coding lifetime? Do users follow some position evolution that other users also follow? Are there a few distinct evolutionary patterns that stand out? Do users program in the same language throughout their coding lifetime? These and many other questions can be answered for SourceForge with the SRDA data and the new CVS data.

4.1.3 Social Network Evolution

With the ability to create social network graphs for any point in time since the inception of these hosting sites, we can evaluate how the graph changes over time. Bonds can be created, strengthened, and broken. Cliques can evolve similarly. Furthermore, network-wide statistics can be examined over a time period which may tell us something important about network evolution.

4.1.4 Economics

The graphs in figures 3.3 and 3.4, there is a lull in activity before the projects become much more active. Initial inspection shows this occurs around the time of the dot-com bubble burst. With significant amounts of data spanning many decades, a study can be done to determine the correlation of the US and world economies.
with OSS project development. Does project activity go up when unemployment rises? How does OSS activity correspond with economic downturns specifically in the IT industry? Data will have to be normalized to adjust for normal growth in OSS projects. In [56], Tsichritzis notes that the dot-com bubble left us with capable and experienced computer users and developers. While many factors affect OSS project contributions, we may be able to determine the effect the bubble had on the OSS community.

4.1.5 Commit Classification

By analyzing the commit information, can we classify the change into categories such as core contribution, bug fix, and new feature? Many researchers have attempted to categorize changes as bug fixes. Weigerber and Diehl in [66] identify refactoring based on code changes. Detecting refactoring based solely on CVS or SVN metadata will likely perform less accurately than analyzing source code, however it may have a unique signature in the logs and warrants some investigation. Obtaining source code from these forges is a long-term goal and a refactoring study on a large scale would likely be very insightful. It may be possible to determine which files within a project are connected in interesting ways based on co-commits as indicated by CVS/SVN logs (similar to that done in [72]). A study towards this goal was done by Zimmerman using source code [73] however patterns indicating connectedness will likely emerge solely from log information. A similar study was done by Bouktif, et al in [6].

4.1.6 Compare OSS Networks

The GNU Savannah at first glance appears to be more tight-knit than SourceForge. SourceForge has a short heavy tail of developers with very little or no activity. SourceForge is more popular and has far more registered users, although its history
is not as long as the data contained in CVS logs for Savannah. Many metrics can be used to examine these networks for comparison. Do Savannah and SourceForge differ in a fundamental way? Xu reported the scale-free nature [3] of SourceForge in [67]. Are BerliOS and Savannah also scale-free networks?

4.1.7 Paid vs. Unpaid

Certain OSS projects are known to have paid contributors, either from donations or salaried workers. How do development activity patterns and management differ between such projects and projects that are strictly volunteer efforts? The CVS/SVN data would allow such analysis to be done with the addition of knowing which projects have paid contributors.
APPENDIX A

VERSIONING DATABASE WORKFLOW

This appendix contains screenshots for the workflow of using the Versioning Database.

Figure A.1. A user is presented with a simple query form shown above.
Users who have contributed to 3dIdf:

gmanndsu
gfinst01

Distinct Commit Times:

<table>
<thead>
<tr>
<th>Time</th>
<th>Cumulative net line changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fri Jul 9</td>
<td>11:09:50 2004 - 0</td>
</tr>
<tr>
<td>Fri Jul 9</td>
<td>12:06:43 2004 - 2</td>
</tr>
<tr>
<td>Fri Jul 9</td>
<td>12:23:03 2004 - 2</td>
</tr>
<tr>
<td>Fri Jul 9</td>
<td>12:43:01 2004 - 2</td>
</tr>
<tr>
<td>Sun Jul 18</td>
<td>17:36:32 2004 - 2</td>
</tr>
<tr>
<td>Mon Jul 19</td>
<td>11:08:30 2004 - 17850</td>
</tr>
<tr>
<td>Wed Jul 21</td>
<td>10:35:04 2004 - 17850</td>
</tr>
<tr>
<td>Wed Jul 21</td>
<td>12:01:52 2004 - 17850</td>
</tr>
<tr>
<td>Wed Jul 21</td>
<td>16:20:54 2004 - 17860</td>
</tr>
<tr>
<td>Thu Jul 22</td>
<td>14:27:51 2004 - 17910</td>
</tr>
<tr>
<td>Fri Jul 23</td>
<td>12:51:00 2004 - 17910</td>
</tr>
<tr>
<td>Fri Jul 23</td>
<td>13:07:48 2004 - 17917</td>
</tr>
<tr>
<td>Tue Jul 27</td>
<td>14:42:46 2004 - 17940</td>
</tr>
<tr>
<td>Wed Jul 28</td>
<td>15:15:05 2004 - 17957</td>
</tr>
<tr>
<td>Sat Aug 7</td>
<td>03:45:43 2004 - 17957</td>
</tr>
<tr>
<td>Sun Aug 8</td>
<td>17:19:29 2004 - 17957</td>
</tr>
<tr>
<td>Thu Aug 12</td>
<td>16:25:19 2004 - 17957</td>
</tr>
<tr>
<td>Thu Aug 12</td>
<td>16:35:32 2004 - 17963</td>
</tr>
<tr>
<td>Fri Aug 13</td>
<td>11:32:45 2004 - 17968</td>
</tr>
<tr>
<td>Mon Aug 16</td>
<td>12:36:15 2004 - 17968</td>
</tr>
<tr>
<td>Tue Aug 17</td>
<td>15:38:33 2004 - 17990</td>
</tr>
<tr>
<td>Wed Aug 18</td>
<td>15:03:41 2004 - 17990</td>
</tr>
<tr>
<td>Fri Aug 20</td>
<td>12:16:21 2004 - 18011</td>
</tr>
<tr>
<td>Mon Aug 23</td>
<td>15:11:54 2004 - 18028</td>
</tr>
<tr>
<td>Mon Aug 23</td>
<td>17:22:45 2004 - 18035</td>
</tr>
<tr>
<td>Tue Aug 24</td>
<td>16:41:50 2004 - 18046</td>
</tr>
<tr>
<td>Thu Aug 26</td>
<td>15:28:29 2004 - 18073</td>
</tr>
<tr>
<td>Thu Aug 28</td>
<td>16:02:57 2004 - 18075</td>
</tr>
</tbody>
</table>

Figure A.2. A list of users who have contributed to that project is displayed along with a list of distinct commits (MRs) and the cumulative number of line changes for that MR. All displayed information is a link to further details about either that person's work on this project or the view of all files and comments made for a particular MR.
Figure A.3. A graph similar to the above is automatically created for the queried project along with additional descriptive information displayed below it.
Figure A.4. This graph is also automatically created. Below that is a list of all files in the file tree. Each file name is a link to the full CVS history of that particular file.
Figure A.5. The full history of a user on a project is displayed. This historical information is that which appears in the CVS logs, however most information is linked so that further data can be browsed regarding a particular MR or file.
Figure A.6. The full revision history is displayed for a particular file. The data is linked similarly to the previous figure.
Figure A.7. A modification request is displayed. 4 files were committed simultaneously by one user. Such a group of simultaneous commits cannot be displayed by cvs tools alone, but is often of interest to researchers.
The following series of screenshots displays a typical workflow of a researcher using SRDA.

Figure B.1. A researcher may start at the schema browser to find information on a particular schema and table.
Figure B.2. A user can type in a schema or click on a listed schema to get to this page which lists all the tables in that schema.

Figure B.3. A user can then click on a particular table to get detailed information on the fields in that table.
Figure B.4. Once a user knows what table and field(s) to query, they can enter that query on the query form page shown above.

Figure B.5. While the query is executing, the above page will be displayed.
Figure B.6. When the query is finished, a message will be displayed indicating such and a link to the results is provided.
Figure B.7. A user has a choice between xml and text results. Shown above is a sample xml result set displayed in a browser.
Introduction

SourceForge.net (http://sourceforge.net/) is the world’s largest Open Source software development website, with the largest repository of Open Source code and applications available on the Internet. Owned and operated by OSTG, Inc. (OSTG), SourceForge.net provides free services to Open Source developers. This website is database driven and the supporting database includes historic and status statistics on projects and user activities at the website. OSTG has shared and intends to continue sharing certain SourceForge.net data (the Data) with the University of Notre Dame (UND) for the sole purpose of supporting academic and scholarly research on the Open Source Software phenomenon. OSTG has given UND permission to in turn share Data with other academic researchers studying the Open Source software phenomenon; provided, however, that UND may not provide all of the Data to any one requesting party without OSTGs prior written consent.

Release of Data

To advance the understanding of Open Source software phenomenon, all or portions of the Data will be made available to researchers (Researchers) on a case-by-case basis in accordance with these provisions and an underlying agreement between OSTG and UND. All requests for all or portions of the Data must be submitted in writing to the UND Principal Investigator by the Researcher. Only parties recognized by

72
OSTG and the UND Principal Investigator as academic and scholarly researchers are eligible to receive the Data. To receive all or a portion of the Data in accordance with these provisions, Researcher must sign this document, agree to abide by SourceForge.net's Terms and Conditions of Use (the Terms of Use), available at http://sourceforge.net/docman/display_doc.php?docid=6048&group_id=1, and observe the additional restrictions listed below (collectively, Restrictions). In addition to other possible remedies, failure to observe these Restrictions may result in revocation of permission to use the Data as well as denial of access to additional Data distributed by OSTG and UND. Data may be made available to Researcher via an Internet site or on CD or other media. There will be no charge for Data made available and downloaded via the Internet.

**Consent**

In addition to agreeing to abide by SourceForge.net's Terms of Use, Researcher agrees to the following additional restrictions (the Additional Restrictions):

1. **Redistribution:** Without prior approval from the UND Principal Investigator, Data, in whole or in part, will not be further distributed, published, copied, or disseminated in any way or form whatsoever, whether for profit or not. This includes further distributing, copying or disseminating to a different facility or organizational unit within the Researchers university, organization, and/or company, as applicable.

2. **Modification and Commercial Use:** Data, in whole or in part, may not be modified or used for commercial purposes. The license granted herein is specifically for the internal research purposes of Researcher, and Researcher shall not duplicate or use Data, or any seal, logo, mark, or phrase associated with or owned by UND and/or OSTG, to manufacture, promote, or sell products or technologies (or portions thereof) either directly or indirectly for commercialization or any other direct for-profit purpose without the prior written permission of UND.
3. Requests for Data from the SourceForge.net database: All requests for Data from the SourceForge.net database will be forwarded to the UND Principal Investigator.

4. Publication Requirements: Those seeking to include more than 2 kilobytes of actual Data from the SourceForge.net database in reports, papers, and other documents to be published or released must first obtain approval in writing from the UND Principal Investigator. Data shall not be used in any way that could cause SourceForge.net users, UND, or OSTG embarrassment, mental anguish, or other harm. In all cases, project ID, user ID data, and other identifying attributes shall be anonymized to preserve privacy of the SourceForge.net users.

5. Citation: All documents and papers that report on research that uses SourceForge.net Data must acknowledge the use of the database by including an appropriate citation that will be provided upon request by the UND Principal Investigator.

6. Publications to UND: A copy of all reports and papers that are for public or general release that use the SourceForge.net database must be forwarded immediately upon release or publication to the UND Principal Investigator.

7. Conflicts. Any conflict between these Additional Restrictions and SourceForge.net’s Terms of Use shall be resolved in favor of these Additional Restrictions.

8. Indemnification: Researcher and/or researchers employing institution agree to indemnify, defend, and hold harmless SourceForge.net, OSTG, Inc., the University of Notre Dame du Lac and its Board of Trustees, officers, employees and agents, individually and collectively, from any and all losses, expenses, damages, demands and/or claims based upon any such injury or damage (real or alleged) and shall pay all damages, claims, judgments or expenses resulting from Researchers use of the SourceForge.net database.
BIBLIOGRAPHY


