Mining CVS repositories, the softChange experience

Daniel M. German
Software Engineering Group
Department of Computer Science
University of Victoria
dmgerman@uvic.ca

Abstract

CVS logs are a rich source of software trails (information left behind by the contributors to the development process, usually in the forms of logs). This paper describes how softChange extracts these trails, and enhances them. This paper also addresses some challenges that CVS fact extraction poses to researchers.

1. Introduction

We have defined *software trails* as information left behind by the contributors to the development process, such as mailing lists, Web sites, version control logs, software releases, documentation, and the source code [5]. Software trails maintain a history of the development that can be used to recover the evolution of the project, to help management understand how it evolves and how its contributors work and interact, and to assist its contributors in their daily tasks.

In particular, software configuration management software, and more specifically version control software, keeps the complete history of any file in the project, including who modified what, when, and the delta of the modification. CVS, the Concurrent Versions System, is arguably the most widely used version control management system available in the market and has become a de-facto standard in the development of open source projects. softChange is a tool for the extraction, enhancement and visualization of software trails, primarily from CVS. The architecture of softChange is depicted in figure 1. The *trails extractor* is responsible for retrieving the raw software trails from the different sources. A SQL relational database management system is the core of softChange. A *fact enhancer* analyses the database in order to generate new facts. Finally, the *visualizer* is responsible for showing the trails to the user. softChange has been successfully used to recover the history of the software project Evolution (a mail client for Unix similar to Microsoft Outlook). The results are reported in [5]. softChange was used to extract Evolution's software trails, enhance them, and then query and visualize them. softChange helped us to understand how the project evolved, and how its developers collaborated. Another research project in which softChange was used is described in [4]. In this case we were interested in understanding the way that the software developers of the GNOME project (a large, open source project) collaborated. The analysis of these software trails allowed the discovery of interesting facts about the history of the project: its growth, the interaction between its contributors, the frequency and size of the contributions, and the important milestones in its development.

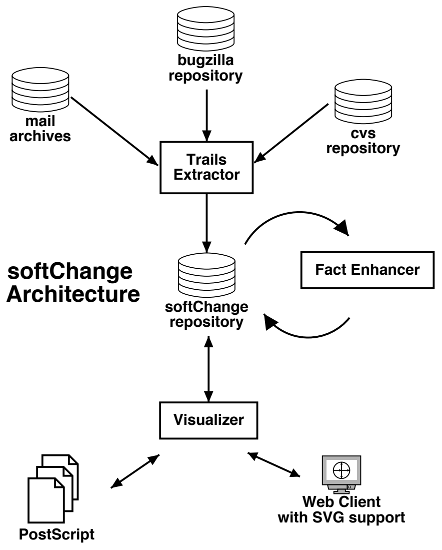

Figure 1. The architecture of softChange

This paper describes how softChange extracts software trails from CVS, and the methods used to create new facts. Section 2 describes related work, section 3 explains softChange's fact extraction in detail, with examples of how it was used to extract the CVS software trails from four major projects. Section 4 describes current challenges in trail extraction from CVS. We finish with our conclusions and future work.

2. Related Work

The two most commonly used hypertext frontends to CVS are Bonsai [7] and lrx [6]. They operate by retrieving the revision information of each file, which is then stored in a relational database. Xia is a plugin for Eclipse for the visualization of CVS repositories[10] based on the Shrimp visualization tool [9]. Xia does not extract the CVS trails, instead it relies on the Eclipse's API to CVS, which makes it extremely slow in large projects. Liu and Stroulia have developed JReflex, a plug-in for Eclipse for instructors of software engineering courses [8]. It is designed to compare the differences in development styles in different teams, who does what, who works on what part of the project, etc. JReflex is intended to be a management oriented tool for browsing the CVS historical data. It uses the history log in CVS and the output of CVS log and stores the information in a relational database. Fisher and Gall have described a CVS fact extractor in [1], where they described the main challenges of creating a database of CVS historical data and then use it to visualize the interrelationships between files in a project [2]. In [3] CVS logs are used to expose relationships between classes and files that might not be found by other methods, such as call graphs.

3. Mining a CVS repository

3.1. Retrieving file revisions

Projects mining CVS historical data have relied on processing the output of its commands (e.g. `cvs log`), or the log files in the repository (`CVSROOT/history`). Unfortunately, the format of the output of CVS commands and its log files is not fully documented. In order to understand these formats, the first phase of the development of softChange used a "clean room" method. Our goal was to recover all the revisions to all the files in the repository. We followed the following procedure:

1. We selected one project as a test case (Evolution), and detailed the requirements for the extractor.
2. We were divided into two independent teams.
3. Each team reversed engineered the CVS formats and proceeded to create the extractor.
4. The extractors were run on the Evolution CVS repository, and their outputs compared.
5. When there were differences in the outputs both teams discussed the problem and determined which team's extractor was faulty (in some case, both were). Teams exchanged information about the formats but there was never exchange of code between teams.
6. We repeated this process until the extractors generated the same output.
7. The code from one team was dropped, and the other became the core of softChange.
8. We then proceeded to create a set of test-cases for future regression testing.

We have used softChange to extract the file revisions from several projects. Table 1 shows the main statistics of four selected projects. A snapshot of their CVS repositories was made on Feb 17, 2004[1]. Mozilla [2] corresponds to the cross-platform Web browser, Evolution is a email client for Unix similar to Outlook, PostgreSQL is a SQL database management system; GNU gcc is the multi platform, multi language compiler.

Table 1. CVS statistics from selected projects

Project	Authors.	Files	Revisions
Mozilla	672	81,520	709,234
Evolution	245	5,402	92,688
PostgreSQL	24	3,789	74,541
GNU gcc	214	24,463	60,311

3.2. Rebuilding modification requests

CVS mining projects usually work at the file revision level. Files, however, are not usually modified alone. A developer will modify all the files necessary to complete a given task, and then commit them together (using the `cvs commit` command). Knowing which files are modified at the same time is important because it means that these files are somehow related (the change in one file is related to the change in the other file).

[1]You can find a copy of the cvs log command for each of the reviewed projects in http://view.cs.uvic.ca/softChange/mining2004/

[2]The Mozilla CVS repository keeps track of the email address of the developer in its cvs id. A typical Mozilla cvs id has the form userid%domainname. An inspection of the different cvs ids suggests that the same developer has used different cvs ids, as her corresponding email address changes. For example these are three different cvs ids that seem to correspond to the same person: `alecf`, `alecf%flett.org`, `alecf%netscape.com`. There are 505 unique cvs ids when the domain name suffix has been stripped.

Unfortunately CVS does not keep track of which files are committed at the same time. By analyzing the files' revisions softChange tries to recover, for each cvs commit, the files that its invocation modified. We denote a modification request (MR) as the set of files committed simultaneously by a developer in a "cvs commit" command.

To our knowledge, softChange is the only tool that attempts to recover modification requests. It uses a heuristic that is based on a sliding window algorithm. This algorithm takes 2 parameters as input: the maximum length of time that an MR can last δ_{max}, and the maximum distance in time between two file revisions τ_{max}. This algorithm is depicted in figure 2. Briefly, a file revision is included in a given MR if a) all the file revisions in the MR and the candidate file revision were created by the same author and have the same log message (a comment added by the developer during the commit); b) the candidate file revision is at most τ_{max}seconds apart from at least one file revision in the MR; and c) the addition of the candidate file revision to the MR keeps the MR at most δ_{max}seconds long.

```
// front(List) removes the front of the list
// top(List) and last(List)
//    query the corresponding elements of the list
// Initialize set of all MRs to empty
$MRS = \emptyset$
**for each** $A$ **in** $Authors$ **do**
    $List = Revisions\ by\ A\ ordered\ by\ date$
    **do**
        $MR.list = \{front(List)\}$
        $MR.sTime = time(MR.list_1)$
        **while** $first(List).time - MR.sTime \leq \delta_{max} \wedge$
                $first(List).time -$
                    $last(MR.list).time \leq \tau_{max} \wedge$
                $first(List).log = last(MR.list).log \wedge$
                $first(List).file \notin MR.list$ **do**
                $queue(MR.list, front(List))$
        **od**
        $MRS = MRS \bigcup \{MR\}$
    **until** $List \neq \emptyset$
**od**
```

Figure 2. Algorithm to recover MRs

Most MRs take few seconds to complete. But some tend to be rather long. There are several factors that affect the length of a MR. First, the size and number of files that compose the MR; second, the bandwidth available between the developer's computer and the CVS server (a slow link will slow down the time required to do the commit); and third, the load of the CVS server. In our experiments we have found that $\tau_{max} = 45s$ and $\delta_{max} = 600s$ are good values for these parameters (these values were used to extract the MRs discussed in this paper). Smaller values for these parameters tend to split MRs, and larger numbers tend to combine two MRs into one).

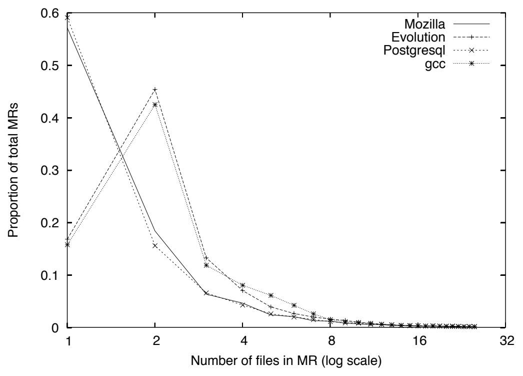

Figure 3. Number of files in an MRs

Most MRs contain very few files. Figure 3 shows the distribution of the number of files in a MR (normalized to values from 0 to 1). The plot only shows MRs with 25 or less files), but there are larger MRs (for example, in Evolution we detected an MR which included 650 files, and in Mozilla one that included 5838 files). Note that the four projects have only 2, almost identical curves. This effect was interesting enough to further explore. We discovered that the use of *ChangeLog* files (files that document the changes made to the software) accounted for this sharp difference. Evolution and GNU gcc use *ChangeLogs*, and almost every MR that includes two or more files includes a change to a *ChangeLog* file. Mozilla and PostgreSQL do not use them. When *ChangeLogs* are not taken into account all the curves look remarkably similar. Further research is needed to verify if this is a coincidence or, indeed, this is a normal pattern in software development. Figure 4 shows the distribution of MRs during 2003 for the chosen projects.

3.3. Other software trails

softChange is able to retrieve and use other trails:

- *ChangeLog* files. If the project uses *ChangeLogs*, for every MR softChange extracts the delta of the corresponding *ChangeLog* file and associates it with it. *ChangeLogs* were originally defined by the Free Software Foundation, and they are commonly found in open source projects, and their objective is to explain how earlier versions of software are different from the current version. Figure 5 shows an excerpt of a *ChangeLog*. The format of a *ChangeLog* delta is very simple: the first line contains the date and author, followed by a sequence of changes (all indented).

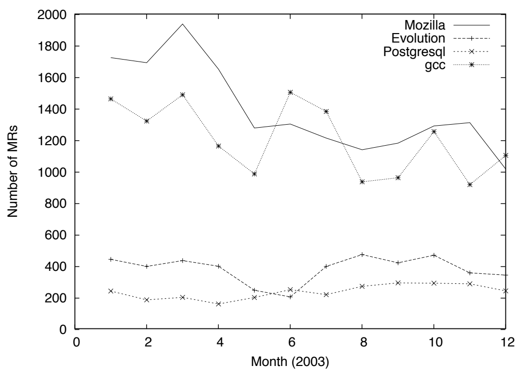

Figure 4. MRs per month, 2003

ChangeLogs are usually created by hand, although there are some utilities that help the developer in their creation. We plan to compare the information found in MRs with the one recorded in *ChangeLogs* in order to verify, both, the MR extraction algorithms and the quality of the *ChangeLogs*. Some projects use the *ChangeLog* delta as the corresponding CVS log message for a given MR.

- Bugzilla: It is customary for developers to record the Bugzilla bug number in the corresponding CVS log message of the MR that fixes it. Because this is a free-form, textual field, there is no standard on how this information should be recorded. Figure 6 shows several cvs log comments that correspond to bug fixes. Based on our observations, the following regular expression matches the most commonly forms in which a bug number is reported ($\backslash s$ corresponds to any white space character):

  ```
  (\#[0-9][0-9]+|bugs?\s+\#?[0-9][0-9]+)(,\s+\#[0-9][0-9]+)*
  ```

  Unfortunately this is an error prone approach and the bug numbers identified need to be correlated to the Bugzilla database, in order to find out if the time that the MR was committed is consistent with a change in the bug report (softChange does not currently support this verification).

- Mailing lists. Mailing lists are an important source of information about the evolution of the project. We currently correlate MRs to mail messages by using the author and the date attributes of both the MR and the message. One of the problems we have encounter mining email messages is that a person tends to have multiple email address, which might not be the same as the ones recorded in the *ChangeLogs*.

```
2003-01-27  Ettore Perazzoli  <ettore@ximian.com>

        * tools/evolution-addressbook-export.c: #include bonobo-activation
        instead of oaf.
        (main): Initialize using gnome_program_init().
        (save_cards): Use g_main_loop_quit() instead of gtk_exit().

        * tools/evolution-addressbook-import.c: Update include list for
        GNOME 2.
        (main): Initialize using gnome_program_init().
        (unref_executable): Use g_main_loop_quit() instead of gtk_exit().
        (add_cb): Likewise.
```

Figure 5. Excerpt from a *ChangeLog*

```
...
       * mail-display.c (mail_display_render): Set default text color
       as black in body when doing printing preview. Fixs bug #48290.
...
Bugzilla bug #218: define HAVE_STRERROR only if it is not yet defined.
Thanks to David Nebinger (dnebinger@synertech.highmark.com) for reporting
the problem and suggesting the fix.
```

Figure 6. Excerpts from cvs log comments referring to bug reports.

4. The challenges of mining CVS repositories

One of the consequences of a growing number of projects extracting different software trails is the explosion of terms to describe them. For example, [1] refers to a CVS revision as a *cvsitem*, and its log as *description*. As the field of mining software repositories matures, we expect that the nomenclature becomes more consistent.

CVS commands are able to access a CVS repository in two different forms: across the network (when the CVS server is located on a different computer), or in the local file system (when the CVS repository is located on the same computer as the working copy of the repository). Mining the repository might result in numerous requests and a large amount of resulting data. For example, softChange can regenerate every revision of a file. In a project such as Mozilla, this will require requesting more than 0.7 million different files (one for each filename, revision pair). If the repository is located in a different computer, this process will most certainly stressed it, and it will consume a large chunk of its bandwidth. This problem will be aggravated if several researchers start using Mozilla as a test case. This problem can be avoided by having a local snapshot of the project's CVS repository. Having a local copy of the repository will guarantee that the resources of the software project are not over-used.

The creation of a common set of test cases is also needed. Different research groups analyzing different software trails have chosen different applications for their analysis. This makes it difficult to compare approaches. We propose the selection of a small set of application that could be used for this purpose. The applications should satisfy the following requirements:

- These applications should be a mixture of old and new, applications, large, medium sized and maybe small ones. Some should include a GUI, while some should not have any. Some might be dead projects.

- Ideally, the original logs and historical data should be made available to the researchers. For instance, the researchers should have a copy of the CVS repository, a dump of the Bugzilla data, a copy of the raw mailing lists archives, etc. This is important because it avoids potential problems created by extracting the data from views of it (such as scrapping bugzilla data from its Web front-end) and it also avoids the extra load on the project servers due to the requests made by the research project.

- It is necessary to agree on the period of observation of the project. Most likely, the chosen projects are alive, and keep changing. Hence it is necessary to specify the start and end date for the observation of a given project.

- Projects with open source licenses are desirable. It is important that the project being analyzed does not put any restrictions on the researcher (like not being able to publish benchmarks of the application). An open source license guarantees no discrimination against using the software. It also provides access to the source code, and equally important, to its software trails. It is undeniable that close-source applications are worth exploring, but they cannot be used as test-cases because they might not be available to any researcher that wants to look at them.

Some projects have become typical test cases in the literature. Mozilla, for example, is one of them. But one has to understand the characteristics of a project before using it as a test case, in order to interpret its data correctly. The Mozilla project started using CVS when the source code of Netscape became Mozilla, and therefore, not all its history is recorded. Another peculiar feature of Mozilla is that several developers have more than one CVS id (we have found developers with two or three cvs ids). Nonetheless, it is a very valuable test case, as it provides the researcher with a large, mature and widely used project that keeps evolving and it is maintained by a large number of individuals.

One important issue that has not been clearly addressed yet is the ethical one. Would the developers of an open source project consider their software trails open too? What are the implications of publishing aggregated data about a project? For example, would it be ethical to claim (in a research paper for example) that code from certain developer tends to have more defects than any other developer's code in the same project? Should projects and their developers be anonymized? The answers to these questions could be the subject of an entire paper.

5. Conclusions and Future Work

CVS is widely used in software projects, some of which are several years old. The information available in its logs can be very valuable for its developers, their management and researchers as it provides a fine-grained view of how the software project is evolving. Unfortunately the amount of data can be overwhelming. Work is needed in several directions: models to describe this data, and query and visualization tools to inspect it. softChange is still under development, but we welcome people interested on using it.

Acknowledgments

This research was supported by NSERC Canada, and the Advanced Systems Institute of British Columbia. The author would like to thank A. Mockus (original co-author of `softChange`) and the anonymous reviewers of this paper.

References

[1] M. Fischer, M. Pinzger, and H. Gall. Populating a Release History Database from Version Control and Bug Tracking Systems. In *Proceedings of the International Conference on Software Maintenance*, pages 23–32. IEEE Computer Society Press, September 2003.

[2] M. Fisher and H. Gall. MDS-Views: Visualizing problem report data of large scale software using multidimensional scaling. In *Proceedings of the International Workshop on Evolution of Large-scale Industrial Software Applications (ELISA)*, September 2003.

[3] H. Gall, M. Jazayeri, and J. Krajewski. CVS Release History Data for Detecting Logical Couplings. In *Proc. of the International Workshop on Principles of Software Evolution (IWPSE)*, pages 12–23. IEEE Press, November 2003.

[4] D. M. German. Decentralized open source global software development, the GNOME experience. *Journal of Software Process: Improvement and Practice*, accepted for publication.

[5] D. M. German. Using software trails to rebuild the evolution of software. *Journal of Software Maintenance and Evolution: Research and Practice*, to appear, 2004.

[6] A. G. Gleditsch and P. K. Gjermshus. lrx Cross-Referencing Linux. http://lxr.sourceforge.net/, Visited Feb. 2004.

[7] T. Hernandez. The Bonsai Project. http://www.mozilla.org/projects/bonsai, Visited Feb. 2004.

[8] Y. Liu and E. Stroulia. Reverse Engineering the Process of Small Novice Software Teams. In *Proc. 10th Working Conference on Reverse Engineering*, pages 102–112. IEEE Press, November 2003.

[9] M.-A. D. Storey, C. Best, and J. Michaud. SHriMP Views: An Interactive and Customizable Environment for Software Exploration. In *Proc. of International Workshop on Program Comprehension*, May 2001.

[10] X. Wu. Visualization of version control information. Master's thesis, University of Victoria, 2003.