Visualizing the evolution of software using softChange

Daniel M. German, Abram Hindle and Norman Jordan
Software Engineering Group
Department of Computer Science
University of Victoria
{dmgerman,abez,njordan}@uvic.ca

Abstract

A typical software development team leaves behind a large amount of information. This information takes different forms, such as mail messages, software releases, version control logs, defect reports, etc. softChange *is a tool that retrieves this information, analysis and enhances it by finding new relationships amongst it, and allows users to to navigate and visualize this information. The main objective of* softChange *it to help programmers, their management and software evolution researchers in understanding how a software product has evolved since its conception.*

Keywords Software evolution, software trails, CVS, visualization, softChange.

1. Introduction

Many software projects use a version control repository to record the the evolution of their source code. These repositories keep track of every change to any source file of the project, including metadata about the change, such as author and date when it happened. Over time, the amount of revisions to a project become enormous. For example, the Mozilla project is composed of 35,000 files which have been modified 450,000 times in 5.5 years of development (from March 1998 to Aug. 2003) by 500 different developers.

CVS, the Concurrent Versioning 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.

While CVS is a very powerful tool, it provides many barriers to the extraction and visualization of valuable information. CVS commands are cryptic and often tell you exactly what your query asked and nothing more. CVS queries often produce an excess of information which is hard for the frustrated developer to sift through. General summaries are rarely provided. CVS does not provide an alternative to browsing through its information.

CVS is built around a group of command-line programs. Several GUI applications have been built around (winCVS, tkCVS, cvsWeb, LinCVS, Pharmacy, gCVS, etc) and some integrated development environments (such as Eclipse) provide a GUI to CVS. In all these cases, the tools are created around the CVS commands and options, providing nothing more than a fancy GUI to the actual commands.

One of the main disadvantages of CVS is that it is not transaction oriented. In other words, when a developer proceeds to "commit" a group of changes to a number of files, CVS does not keep track of all the files modified by this commit operation. It treats each change to a file independently of the other files included in the commit. After the commit has taken place, CVS does not know which files were modified together. This information, however, is important because it highlights coupling amongst files: if two files are modified at the same time, it is because they share *something* in common. We refer in this paper to an commit operation as a *modification request* (MR). A MR is therefore a collection of revisions to files that are modified at the same time.

The information stored in the CVS repository is quite valuable as it can help answering many questions. It can assist developers in knowing who has modified which files and when. It can help the administration in trying to understand the modification patterns of the project and the way the different team members interact. Finally, it can help in the recovery of the evolution of the project [5]. For example, developers can ask the following questions [10]:

- What happened since I last worked on this project?
- Who made this happen?
- When did the change take place?
- Where did the change happen?

- Why were these changes made?
- How has the file changed?
- What methods or functions were changed?
- What is the frequency of change?
- What files changed?
- Who is working on each modules?

Administrators, on the other hand, are interested in higher level questions and metrics such as:

- How often does a programmer complete a MR?
- How much does the programmer change per MR
- What kind of commits does one programmer do?
- How much changed between each release?
- How many bugs are fixed and found after a stable release?
- What kind of modifications are done at a certain time?
- When was a module stabilized?
- What is the daily LOC count for each programmer?
- When is a module actively being developed and maintained?

We define *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]. In this paper we describe softChange, a tool that mines software trails from CVS repositories, then enhances this data with some heuristics in order to recover higher level information, such as rebuilding MRs. Each MR is analyzed in order to know what type of changes took place; such as adding new functions, reorganizing source code, adding comments to the code only, etc. After extraction and analysis, softChange provides a graphical and hypertext representation of this information. This paper is divided as follows: previous work is described in section 2; section 3 describes the architecture of softChange; section 4 describes the visualization features of softChange; we end describing our experiences using softChange, our conclusions, and future work.

2. Previous Work

The two most commonly used hypertext front ends to CVS are Bonsai [7] and lrx [6]. They provide a Web interface to the CVS repository and isolate the user from the complexities of the CVS commands (the man page of CVS is 9000 words long, approximately 3 times the length of this paper). Both tools allow the user to inspect the history of any given file in the project and neither of them attempts to enhance the software trails available in the repository.

Xia is a plugin for Eclipse for the visualization of CVS repositories[10]. Xia recovers relations available in the logs of a CVS repository and allows the user to navigate them. It uses squares to represent files, their revisions and developers, and lines to represent the relationships between them. Xia has two main limitations. The first is that Xia relies on the Eclipse API to access the CVS repository. Every time Xia wants to create a view, it queries the CVS repository in order to retrieve the necessary data. This becomes a very expensive operation making Xia extremely slow in large CVS repositories. The second limitation is that Xia operates at the revision level, not at the MR level.

Hipikat aggregates many sources of information such as bugzilla, the CVS repository, mailing lists, emails etc and provides a searchable query interface[1]. The purpose of Hipikat is to ”recommend software artifacts” rather than summarize and visualize them. Thus Hipikat is much like Google for a software project. One interesting feature of Hipikat is that it correlates software trails from different sources, inferring relationships between them.

Liu and Stroulia have developed JReflex, a plug-in for Eclipse for instructors of software engineering courses. JReflex helps the instructor to monitor how different teams of students developed a term project by using their CVS historical information [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. JReflex does not enhance the information available in CVS.

Fisher and Gall have described a CVS fact extractor in [2]. In it they describe the main challenges of creating a database of CVS historical data and then use it to visualize the interrelationships between files in a project [3].

3. softChange Architecture

softChange is composed of four main components, depicted in figure 1.

- Software trails repository: At the core of softChange lies a relational database that is used to store all the historical information.

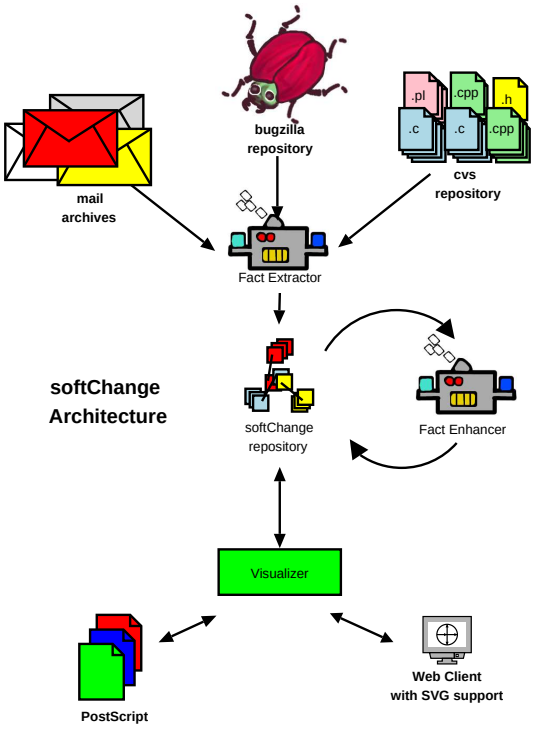


Figure 1. Architecture of AUSS

- Software trails extractor: In a typical software development project, software trails originate from many different sources: CVS historical data, email messages, bug reports, ChangeLogs, etc. The purpose of softChange trails extractor is to retrieve as many software trails as possible. Currently, softChange is able to retrieve trails from CVS, from ChangeLogs, from the releases of the software (the tar files distributed by the software team) and from Bugzilla.

- Software trails analyzer: Once softChange has extracted the software trails, it proceeds to use this information to generate new facts. For example, using a set of heuristics, softChange regroups file revisions into MRs. softChange analyzes the changes in the source code and thus extracts a list of function, methods and classes that have been added, modified or removed from one file revision to the next. softChange also correlates the available software trails; for example, softChange links a given MR to its Bugzilla bug report.

- Visualizer: softChange provides a visualizer to the repository that allows the user to explore the software trails. This front end is described in detail in the next section.

4. Visualizing software trails

One of the main purposes of softChange is to summarize and browse MRs. It will help developers, administrators and researchers explore and understand the development of the project. Instead of tedious typing, a developer or maintainer could quickly navigate through the MRs using the Web visualization front-end of softChange. The visualizer is divided in two main parts: a hypertext browser and a graphical viewer. The hypertext browser is used to navigate through the MRs. Users can choose to navigate MRs by date, by author, or by filename. For each MR, softChange provides the details of what revisions to which files it contains, and any metadata about the modification. The information is cross-referenced so it is possible to navigate amongst any related information by following hyperlinks.

softChange tries to leverage any external sources of information too. One benefit of the hypertext application is the ease of information association. Integration to other existing hypertext tools is quite easy by hyperlinking between tools. If the project provides a bugzilla repository (such as it is the case with many open source projects), a given MR is linked to its corresponding Bugzilla entry. softChange also links to the Bonsai repository of the project if one exists. Figure 4 shows a snapshot of softChange displaying the details of an MR for the Mozilla project.

The graphical viewer of softChange is composed of two main parts. One uses PostScript to generate static plots of the software trails. The other one uses SVG to display the same information more interactively. The SVG version takes advantage of its hypertext capabilities to link points in the plots with their details (by pointing to their details in the hypertext browser of softChange). softChange is able to generate the following plots:

- Growth of LOCS vs time, at the project level and at the module level (a module in softChange is defined as the collection of files under a given subdirectory).

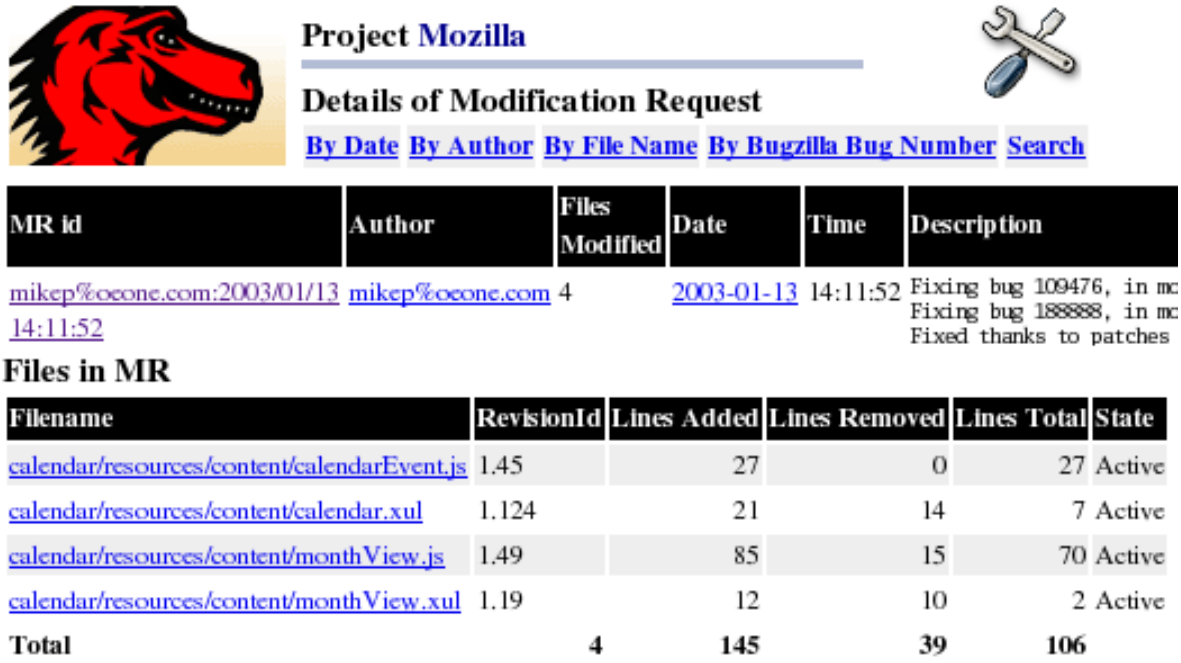
- Number of MRs vs time: How many MRs are committed in a given period?

- Number of files vs time: How many files are part of the project at a given point in time?

- Number of files in a given MR: How many files compose a given MR?

- Proportion of MRs per contributor: What is the distribution of the number of MRs per contributor?

- Proportion of revisions per source code file: How frequently is a given file modified?



Project Mozilla

Details of Modification Request

By Date | By Author | By File Name | By Bugzilla Bug Number | Search

MR id	Author	Files Modified	Date	Time	Description
mikep%oeone.com:2003/01/13 14:11:52	mikep%oeone.com	4	2003-01-13	14:11:52	Fixing bug 109476, in mc Fixing bug 188888, in mc Fixed thanks to patches

Files in MR

Filename	RevisionId	Lines Added	Lines Removed	Lines Total	State
calendar/resources/content/calendarEvent.js	1.45	27	0	27	Active
calendar/resources/content/calendar.xul	1.124	21	14	7	Active
calendar/resources/content/monthView.js	1.49	85	15	70	Active
calendar/resources/content/monthView.xul	1.19	12	10	2	Active
Total	**4**	**145**	**39**	**106**	

Figure 2. Hypertext browser: details of an MR using softChange

- Number of modules that are modified in a given MR: How frequently an MR includes modifications of 2 or more modules?
- Project time-tree: When are given files created and modified, displayed in a timeline fashion?

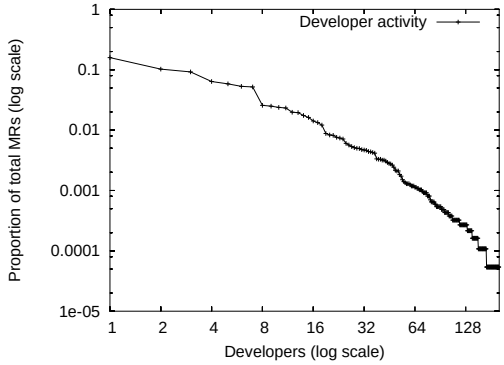


Figure 3. PostScript visualizer: proportion of MRs per contributor.

The PostScript viewer generates plots that are static in nature. The user determines the necessary parameters for the plot, and softChange generates in return an encapsulated PostScript file. Figure 3 depicts the proportion of MRs per developer for Evolution (a GUI mail client for Unix equivalent to Microsoft Outlook). Figure 4 depicts the number of MRs against time in the same project [5].

The SVG viewer takes advantage of the hypertext and interactivity features of SVG. This interaction is highlighted in the project time-tree diagram. The time tree graph is a view of a file directory tree. It depicts the how files populate a given directory (and its child subdirectories) and the proportion of MRs that include them at any point in time; the horizontal axis corresponds to time. Every time a file or directory is created, a new branch is started from the directory line. The user can expand and contract any given subdirectory, in order to avoid information overload. The user is also allowed to zoom-in, and zoom-out in any given region of the plot. Figure 5 depicts this diagram for the project Evolution.

5. Evaluation and Future Work

softChange has been successfully used to recover the history of the software project Evolution. The results are reported in [5]. softChange was used to extract Evolution's software trails, enhance them, and then query and visualize them. The Evolution project was born in 1999. By May 2003, its CVS repository kept track of almost 5000 files, for a total of 77,000 revisions. These revisions were reconstructed into 18,500 different MRs. A total of 201 developers committed at least one revision to the project.

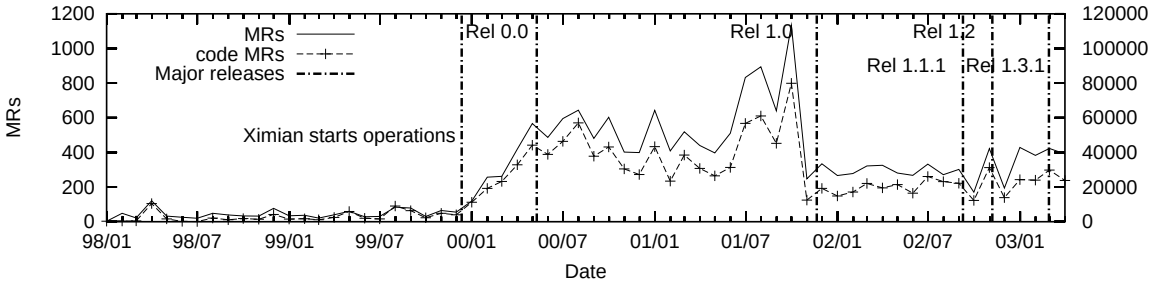


Figure 4. PostScript front-end: MRs over time.

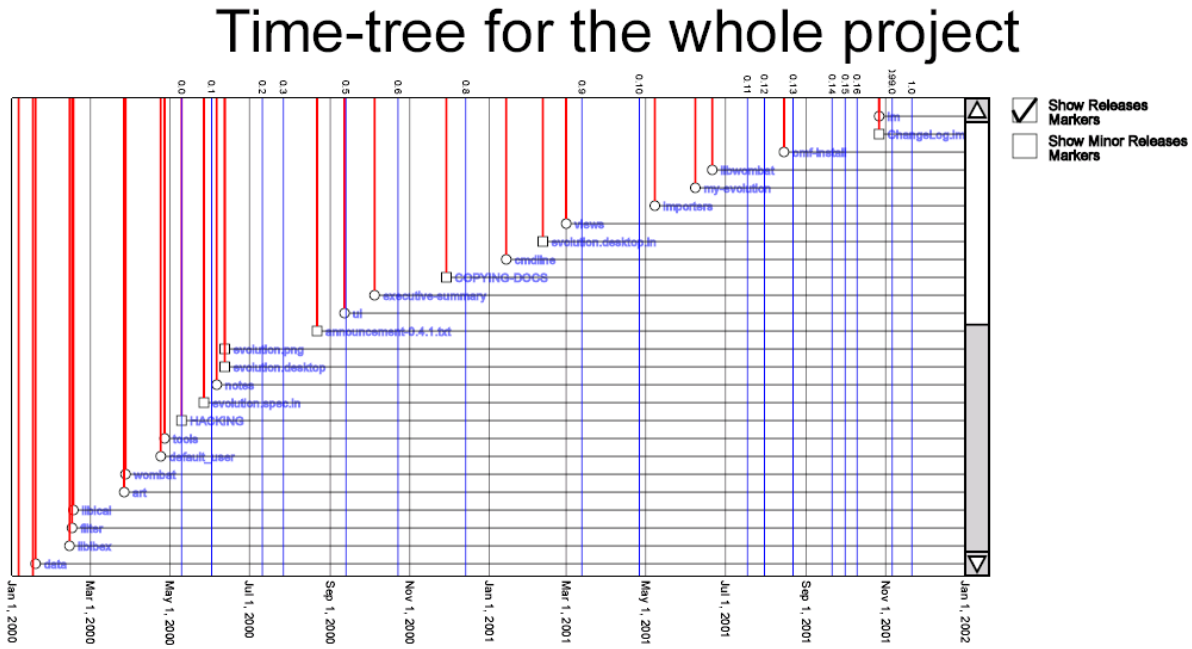


Figure 5. Time-tree in softChange

The size of the historical database created from the software trails of Evolution accounted for approximately 0.5 Gbytes. softChange helps us 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) collaborate. 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.

Given that many of the plots and reports of softChange were designed around the questions described in the introduction, we are confident that softChange is useful to software developers and their management. We expect to maintain the historical data of several projects using softChange and make it available to developers, and then evaluate how they use it.

One of the main advantages of keeping all software trails in a relational database is that we can analyze them and enhance them by extracting new knowledge from them. Our current research is into the characterization of MRs. We want to know what types of MRs are typically committed by developers: are they source code modifications, documentation, or internationalization? If there are changes to the source code, are they bug fixes, new features, reorganization of the code or clean up? With this enhanced information, users can discriminate and select the changes they are interested in, without being overwhelmed by the amount of available data. The more facts that are known about the evolution of the project, the better the visualization tools that can be created.

Data mining of software trails is a promising area. Old, stable software projects have a large amount of software trails available. These trails can be mined for new facts; these facts can be used for better visualizations.

The architecture of softChange permits the use of different visualization tools. We are currently working with JReflex to adapt their Eclipse plug-in to softChange. We are also pursuing using Shrimp [9] to visualize the relationships available in the repository. softChange is an open source project, with an open architecture. We hope that other research projects will help create more trail extraction tools, more fact enhancing algorithms and more visualization tools.

Acknowledgments

This research was supported by the National Sciences and Engineering Research Council of Canada, and the Advanced Systems Institute of British Columbia.

References

[1] D. Cubranic and G. C. Murphy. Hipikat: Recommending pertinent software development artifacts. In *Proceedings of the 2003 International Conference on Software Engineering*, pages 408–418, Portland, May 2003. Association for Computing Machinery.

[2] 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.

[3] 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.

[4] D. M. German. Decentralized open source global software development, the gnome experience. *Journal of Software Process: Improvement and Practice*, Accepted for publication, 2004.

[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.