An Open Framework for CVS Repository Querying, Analysis and Visualization

Lucian Voinea
Technische Universiteit Eindhoven
Postbus 513, 5600 MB Eindhoven
The Netherlands
Tel. +31402474344
l.voinea@tue.nl

Alexandru Telea
Technische Universiteit Eindhoven
Postbus 513, 5600 MB Eindhoven
The Netherlands
Tel. +31402474344
alext@win.tue.nl

ABSTRACT

We present an open framework for visual mining of CVS software repositories. We address three aspects: data extraction, analysis and visualization. We first discuss the challenges of CVS data extraction and storage, and propose a flexible way to deal with CVS implementation inconsistencies. We next present a new technique to enrich the raw data with information about artifacts showing similar evolution. Finally, we propose a visualization backend and show its applicability on industry-size repositories.

Categories and Subject Descriptors

D.2.7 [**Software engineering**]: Distribution, Maintenance, and Enhancement – *documentation, reengineering*; H.3.3 **[Information Storage and Retrieval**]: Information Search and Retrieval – c*lustering, query formulation*; I.3.8 **[Computer Graphics]**: Applications

General Terms

Management, Measurement, Documentation

Keywords

Evolution visualization, software visualization, CVS repositories

1. INTRODUCTION

Software Configuration Management (SCM) systems are proven instruments for managing large software development projects. SCMs maintain a history of changes in the structure and contents of the managed project. This information is very suitable for empirical studies on software evolution.

Many SCM systems exist on the market, e.g. Subversion, Visual SourceSafe, RCS, CMSynergy, ClearCase and CVS. The Concurrent Versions System (CVS), available via the Open Source community, is a very popular SCM system and has been the preferred choice for SCM support in many Open Source projects in the last decade. Many CVS repositories for long evolution periods, e.g. 5-10 years, are freely available for analysis, so CVS is an interesting option for research on software evolution.

However, CVS is mainly designed for archiving data. CVS offers only a basic querying interface for retrieving a given version of a file or an attribute list with the file state evolution. CVS provides no features to let users get data overviews easily. The user feedback, i.e. state attributes list, is provided only in compiled textual format, which makes it unhandy for quick browsing.

Another challenge of CVS-based software evolution research is the data size and complexity. Raw repository information is too large and provides directly just limited insight in the evolution of a software project. Extra analysis is needed to process these data and extract relevant evolution features.

In this paper we address the challenges of software evolution assessment in CVS repositories. We propose an open framework for CVS data extraction and analysis. We illustrate the capabilities of this framework with a customized implementation. The basic questions we try to answer are:

- How to deal with the large size of CVS data and various limitations of textual feedback?
- How to extract logical coupling information from evolution?
- How to efficiently present evolution data to users to enable correlations across entire projects?

The structure of this paper is as follows. In section 2 we review existing CVS data extraction methods and software evolution analysis techniques. Section 3 presents our flexible interface with CVS repositories. Section 4 describes a new clustering technique for detecting logical coupling of files based on evolution similarity. Section 5 describes a visual back-end for evolution assessment and shows it at work on large repositories. Section 6 summarizes our contribution and outlines open issues for future research.

2. BACKGROUND

The huge potential of the data stored in SCMs for empirical studies on software evolution has been recently acknowledged. The growth in popularity and use of SCM systems, e.g. the open source CVS [5] and Subversion [15], opened new ways for

MSR'06, May 22–23, 2006, Shanghai, China.

project accounting, auditing and understanding. These efforts can be grouped in two directions: data mining and data visualization.

Data mining focuses on processing and extracting relevant information from SCM systems. SCM systems have not been designed to support empirical studies, so they often lack direct access to high-level, aggregated evolution information. Hence, information is distilled from the "raw" stored data by data mining tools, as follows. Fischer *et al*. [7] extend the SCM evolution data with information on file merge points. Gall [9] and German [10] use transaction recovery methods based on fixed time windows. Zimmermann and Weißgerber [21] extended this work with sliding windows and facts mined from commit e-mails. Ball analyzes class-cohesion using a mined probability of classes being modified together [1]. Bieman *et al*. [2] and Gall *et al*. [9] also mine relations between classes based on change similarities. Ying *et al*. [20] and Zimmermann *et al*. [21][21] address relations between finer-grained artifacts, e.g. functions. Lopez-Fernandez *et al*. [13] apply general social network analysis methods on SCM data to assess the similarity and development process of large projects.

Data visualization, the second research direction, takes a different path, focusing on making the large mass of evolution information effectively available to users. Visualization methods make few assumptions on the data – the goal is to let users discover patterns and trends rather than coding these in the mining process. SeeSoft [6], a line-based code visualization tool, uses color to show code snippets matching given modification requests. Augur [8] visually combines project artifact and activity data at a given moment. Xia [19] uses treemap layouts for software structure, colored to show evolution metrics, e.g. time and author of last commit and number of changes. Such tools successfully show the structure of software systems and the change dependencies at given moments. Yet, they don't give insight into code attributes and structure changes made throughout an *entire* project. A first step in this direction, UNIX's `gdiff` and Windows' WinDiff display code differences between two versions of a file by showing line insertions, deletions, and edits computed by the `diff` tool. Still, such tools cannot show the evolution of thousands of files and hundreds of versions. To overcome these shortcomings, Collberg *et al*. [4] depicts the evolution of software structures and mechanisms as a sequence of graphs, for medium-size projects. Lanza [12] depicts the evolution of object-oriented software systems at class level. Wu *et al*. [18] visualize the evolution of entire projects at file level and emphasize the evolution moments. Finally, our own work provided software evolution visualizations at several granularity levels: CVSscan [16] for the line-level evolution of a few source code files and CVSgrab [17] for file-level, project-wide evolution investigations.

Data extraction is a less detailed aspect of software evolution analysis. Many works extract data from CVS repositories, e.g [21], [7], [9], [11], [20], [13], [16], and [17]. Yet, a standard framework for CVS data extraction still lacks. Two main challenges exist here: data retrieval and CVS output parsing. The huge amount of data in CVS repositories is usually available over the Internet. On-the-fly retrieval is not suited for interactive assessment, given the sheer data size. Storing data locally requires long acquisition times, large storage space, and consistency checks. Next, CVS output is ill suited for machine reading. Many CVS systems use ambiguous or nonstandard output formats. Attempts to address these problems exist, but are incomplete. Libraries exist that offer an application interface (API) to CVS, e.g. Java's javacvs or Perl's libcvs. However, javacvs is basically undocumented. Libcvs handles only local repositories. The Eclipse environment offers a CVS client implementation but not an API. The Bonsai project [3] offers several tools to populate a database with evolution data obtained from CVS repositories. These tools are mainly meant as a web data access package and are little documented. The best supported effort for CVS data acquisition is the NetBeans javacvs package [14], a well-documented API with allegedly full CVS client support that parses CVS output into API-level data structures. SoftChange [11] was a first attempt for a coherent environment to support the comparison of Open Source projects, targeting CVS, project mailing lists, and bug report databases. It focuses mainly on data extraction and analysis, aiming to be a generic foundation for building evolution visualization tools.

Overall, several tools exist, each addressing different, though overlapping, facets of software evolution analysis (see Table 1).

Table 1: Tools and methods overview

Tool Visualization		Query	Analysis
Libcvs	X		
javacvs	X		
Bonsai [3]	X		
Eclipse CVS plugin	X		
NetBeans.javacvs [14]	X		
Release History Database [7]	X	X	
Diff		X	
WinDiff		X	X
eRose [21]	X	X	
QCR [9]		X	
Social Network Analysis [13]		X	
SeeSoft [6]			X
Augur [8]	X		X
Gevol [4]			X
CodeCrawler [12]			X
Evolution Spectograph [18]		X	X
CVSscan [16]		X	X
CVSgrab [17]		X	X
Xia [19]		X	X
SoftChange [11]	X	X	X

We propose a new approach towards an integrated framework for CVS data extraction, analysis and visualization. Our goal is twofold. First, we aim to provide users with a complete software evolution analysis chain. Secondly, we aim at building an experimenting foundation for research at *all* levels, i.e. extraction, analysis, and visualization. Our approach is described next.

3. CVS QUERYING

CVS data extraction is a main problem for research on software evolution. The CVS Internet protocol unfortunately covers only the main CVS function, i.e. file archiving. The CVS navigation commands do not have a machine-readable output. Navigation feedback is given in a compiled text format that is not always easy to decipher. Often, parse tools for this output fail to work on some repositories due to awkward local conventions, e.g. “date format is yyyy-mm-dd and not dd/mm/yyyy” or “file names may contain spaces”. This makes uniform access to CVS data difficult. In such cases, one usually searches a parser that copes with the output format at hand and tries to add it to the experimental setup. We propose an approach towards CVS data acquisition that simplifies this process using a data acquisition *mediator* (see Figure 1).

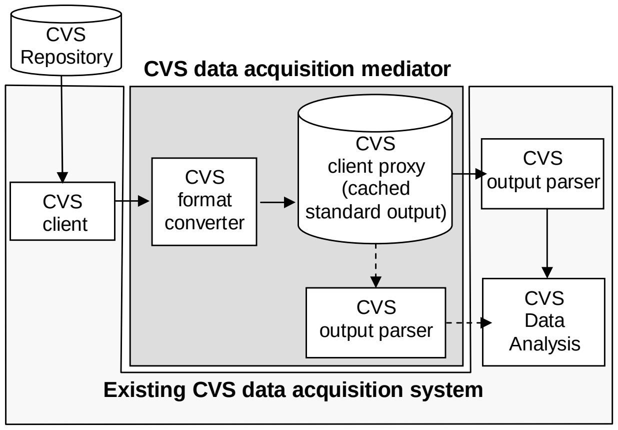
CVS Repository
CVS data acquisition mediator
CVS client
CVS format converter
CVS client proxy (cached standard output)
CVS output parser
CVS output parser
CVS Data Analysis
Existing CVS data acquisition system

Figure 1: CVS data extractor with output format mediator

The mediator is an easy-to-customize preprocessor between CVS repositories and existing data acquisition tools. When format inconsistencies occur between the CVS output and a parser, we don’t need a new CVS data acquisition tool. Instead, we adapt the mediator with a simple rule to transform the new format into the one accepted by the tool. While this doesn’t completely remove the problems of inconsistent output formatting, it is a flexible way to solve problems without removing the preferred data acquisition tool. We developed an open source, easy to customize mediator, in a simple to use programming language: python. Secondly, the mediator provides data access to CVS repositories and can also be easily integrated in projects that lack a data acquisition tool. The mediator offers selective access to CVS repositories, i.e. retrieves only information about a desired folder or file, and also caches the retrieved information locally. This design lets one control the trade-off between latency, bandwidth and storage space in the data acquisition step as desired.

4. DATA ANALYSIS

Raw CVS data is too large and low-level to provide insight in the evolution of software projects. Extra analysis is needed to extract relevant evolution aspects. An interesting analysis use-case is to identify artifacts that have similar evolution. Several approaches exist for this [2], [9], [21], [20]. They all use similarity measures based on recovered CVS transactions, i.e. sets of files committed by a user at some moment. The assumption is that related files have a similar evolution pattern, and thus their revisions will often share the same CVS transaction. This information about correlated files is used to predict future changes in the analyzed system, from the perspective of a given artifact.

We propose a more general approach. We argue that not transactions, but pure commit moments, are important for finding similar files. Transaction-based similarity measures fail to correlate files developed by different authors and with different comments attached, but which are still highly coupled. To handle such cases, we propose a similarity measure using the time distance between commit moments. If $S_1 = \{t_i \mid i = 1..N\}$ are the commit moments for a file F_1 and $S_2 = \{t_j \mid j = 1..M\}$ the commit moments for F_2, we define the similarity between F_1 and F_2 as the symmetric sum:

$$\Phi(F_1, F_2) = \sum_{i=1}^{N} \frac{1}{\sqrt{\min\{|t_i - t_j| \mid t_j \in S_2, |t_i - t_j| < k\} + 1}} + \sum_{j=1}^{M} \frac{1}{\sqrt{\min\{|t_j - t_i| \mid t_i \in S_1, |t_j - t_i| < k\} + 1}}$$

where k is a customizable neighborhood factor intended to reduce the influence of completely unrelated events on the similarity measure. The square root is meant to attenuate the influence of the network latency on the CVS transaction. Intuitively, this measure considers, for each commit moment of F_1, the closest commit moment from F_2, weighted by the inverse time distance between the two moments. We next use this measure in an agglomerative clustering algorithm to group files that have a similar evolution, yielding a logical system decomposition following similar evolution patterns. We make the analysis data available for any evolution assessment back-end by storing it in a flat file database.

5. VISUALIZATION

Visualization tries to give insight in these large and complex CVS data by delegating the pattern detection and correlation making to the human visual system. Visualization can also present the results of data analysis in an intuitive, ready-to-use, way. Visualization is a main ingredient of our CVS repository mining framework.

The data acquisition (Sec. 3) and analysis (Sec. 4) steps are generic and can be used with any visualization back-end. We present now a methodology for quick visual assessment of data analysis results and illustrate its applicability with several use cases. For this, we use the CVSgrab tool, detailed in [17]. CVSgrab visualizes project evolution at file level. It depicts each project as a set of horizontal strips representing files along the time axis (Figure 2).

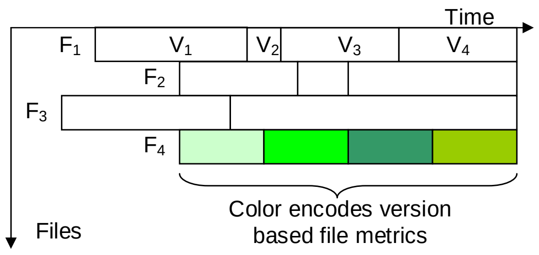

Figure 2: CVSgrab visualization of project evolution

The file layout along the vertical axis is interactively constructed to suit specific analysis needs. Plateau cushions are used to highlight groups of files that have a similar evolution [17]. CVSgrab uses a generic mechanism to map file-level attributes to colors. We next discuss the use of CVSgrab as visualization back-end in our proposed open framework by assessing the evolution of several file metrics on real-life, industry-size CVS repositories. **Error! Reference source not found.** shows the evolution of ArgoUML, an object-oriented design tool with a 6-year evolution of 4452 files developed by 37 authors. To analyze the evolution of ArgoUML using the framework described in this paper, we coupled the CVS data acquisition mediator to the CVSgrab back-end and used the standard CVS client to access the ArgoUML repository over the Internet. Data acquisition took 31 minutes over a T1 Internet connection: 8 minutes for the initial setup (i.e. one-time retrieval of the last version of 56MB) and 23 minutes to retrieve the evolution data to be visualized (29MB).

In **Error! Reference source not found.**, a 12-snapshot matrix shows ArgoUML's evolution. Each column shows the evolution of one metric:

- Column 1 shows the development team evolution. Each file version color shows the ID of the user who committed it.
- Column 2 shows the size evolution of contributions as number of lines. Files that are first committed are colored gray. Blue shows file size increase, red is decrease, and yellow is a commit that affects several lines but does not modify the file size. While hue encodes the type of change in size, brightness encodes the change size: lighter colors denote smaller changes, darker colors denote more modifications.
- Column 3 shows the file type: red for java source files, green for images, and yellow for HTML files.
- Column 4 highlights versions that contain given strings in their associated commit comment: green for versions that contain the word "fix", blue for versions that contain the word "error".

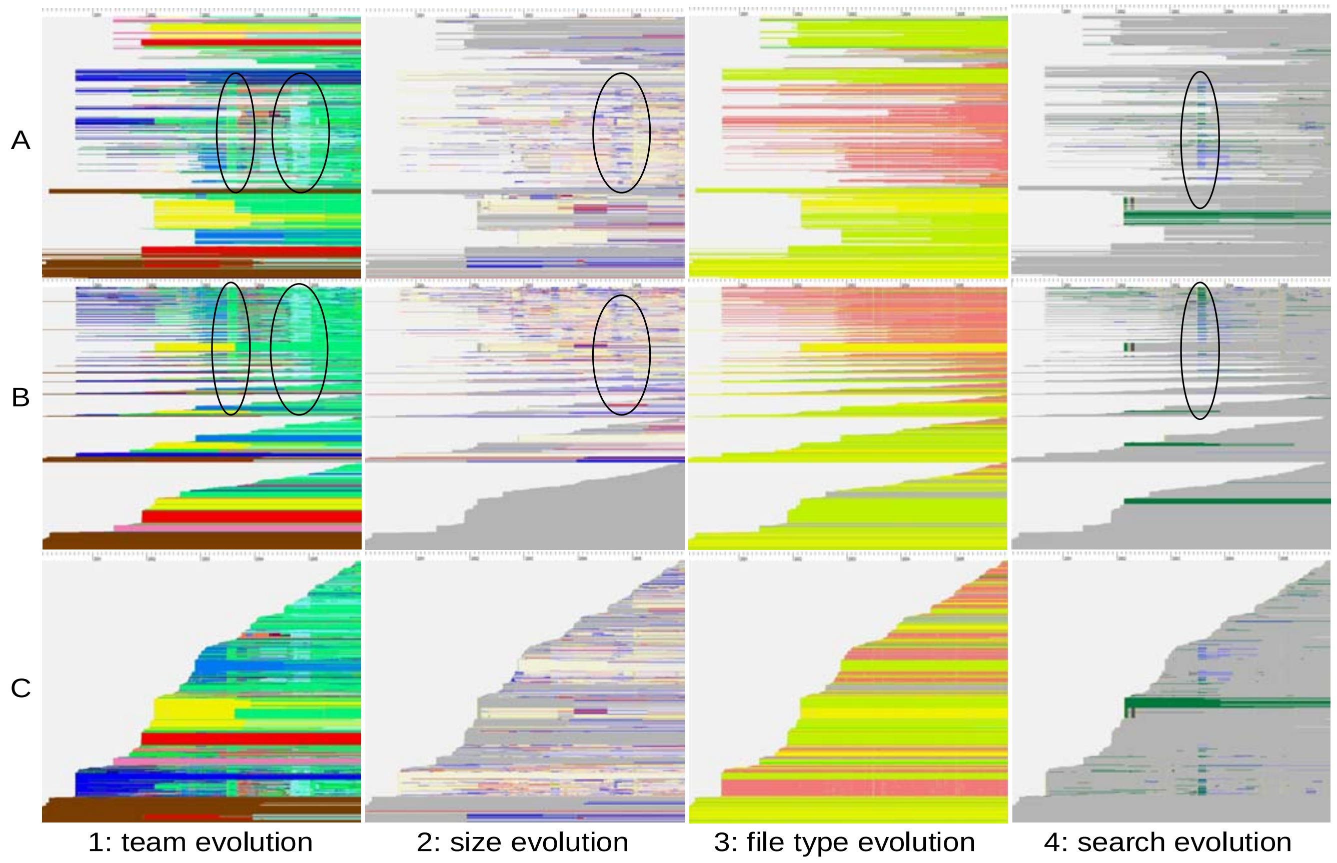

Figure 3: ArgoUML metrics evolution visualization with CVSgrab

Each row in **Error! Reference source not found.** uses another layout offered by CVSgrab. In row A, files are sorted alphabetically on their full path, and thus show the folder structure. In row B, files are sorted from top to bottom in decreasing order of number of versions, i.e. file activity. Files that have the same number of versions are further sorted in decreasing order of creation time. In row C, files are sorted in decreasing order of creation time. Files created in the beginning of the project are at the bottom, while 'young' files are at the top.

By assessing the project evolution shown in **Error! Reference source not found.**, one can discover several interesting aspects of the process and organization of ArgoUML. Cell C3 shows that the project started with a documentation base (i.e. green and yellow) that probably contained the system specification. This was contributed by one user (*jrobbins* = brown in C1) and remained unchanged for the entire project duration except for a large addition (dark blue in C2) done by another user two years later (*dennyd* = red in C1). The added code concerned seemingly an underspecified issue as it was extend again two years later (dark blue in C2) by another author (*mvw* = cyan in C1). The real implementation first appeared 6 months after the specification was committed (java source files = red in C3) and was contributed by one author (*1sturm* = blue in C1). Two years from the project start, another big documentation chunk was added (yellow and green in C3) by one user (*jeremybennett* = yellow in C1). Although both the specification, implementation, and documentation parts appear to be the work of one author each, it is intriguing the fact they were all committed at one time (i.e. not incrementally), by one author, and contained many files, i.e. approx. 400. It is thus possible that these represent the work of more people, which was first checked in by one single person. For the rest of the project, one user has a significant contribution (*linus* = green in C1), with one exception in the fifth project year (*mvw* = cyan in C1). The large oval in A1 shows that *mvw* (cyan in A1) made a significant contribution (dark blue, large oval in A2) to the implementation (red in A3). The same pattern can be recognized following the large ovals in B1 and B2. A3 shows that the project has a very clean organization. The major color groups correspond to the folders *documentation* (green at the top), *src_new* (red at the middle) and *www* (yellow and green at the bottom). From B3, one can see that most activity during the project was related, as expected, to changes in the implementation files (red at the top) followed by changes in the documentation (yellow in the middle) and in the documentation images (green at the bottom). B2 shows that almost one-third of the files added during the project did not change during all six years (since they are grey). Most such files contain documentation (i.e. yellow and green in B3). To this group belongs also the largest part of the previously identified system specification (i.e. brown in B1, by correlation with C1 and C3). Another interesting aspect is shown in the small ovals in A1 and A4. It seems that in the fourth project year *linus* made a significant contribution, not in terms of size (i.e. no significant size change pattern detected in A2) but in terms of code cleaning. Many implementation files (red in A3) containing the words "fix" and "error" in their revision comment have been committed by *linus* to the repository. The same pattern can be seen in row B. The large green (i.e. "fix") horizontal pattern that can be seen in column 4 corresponds to an initial checkout of documentation files. It suggests that previous work has been done in that area without being committed. **Error! Reference source not found.** shows also that almost no significant decrease took place in the project size. One exception, highlighted in C2, shows a size drop for documentation files (yellow in C3) that occurred at the end of the fourth project year.

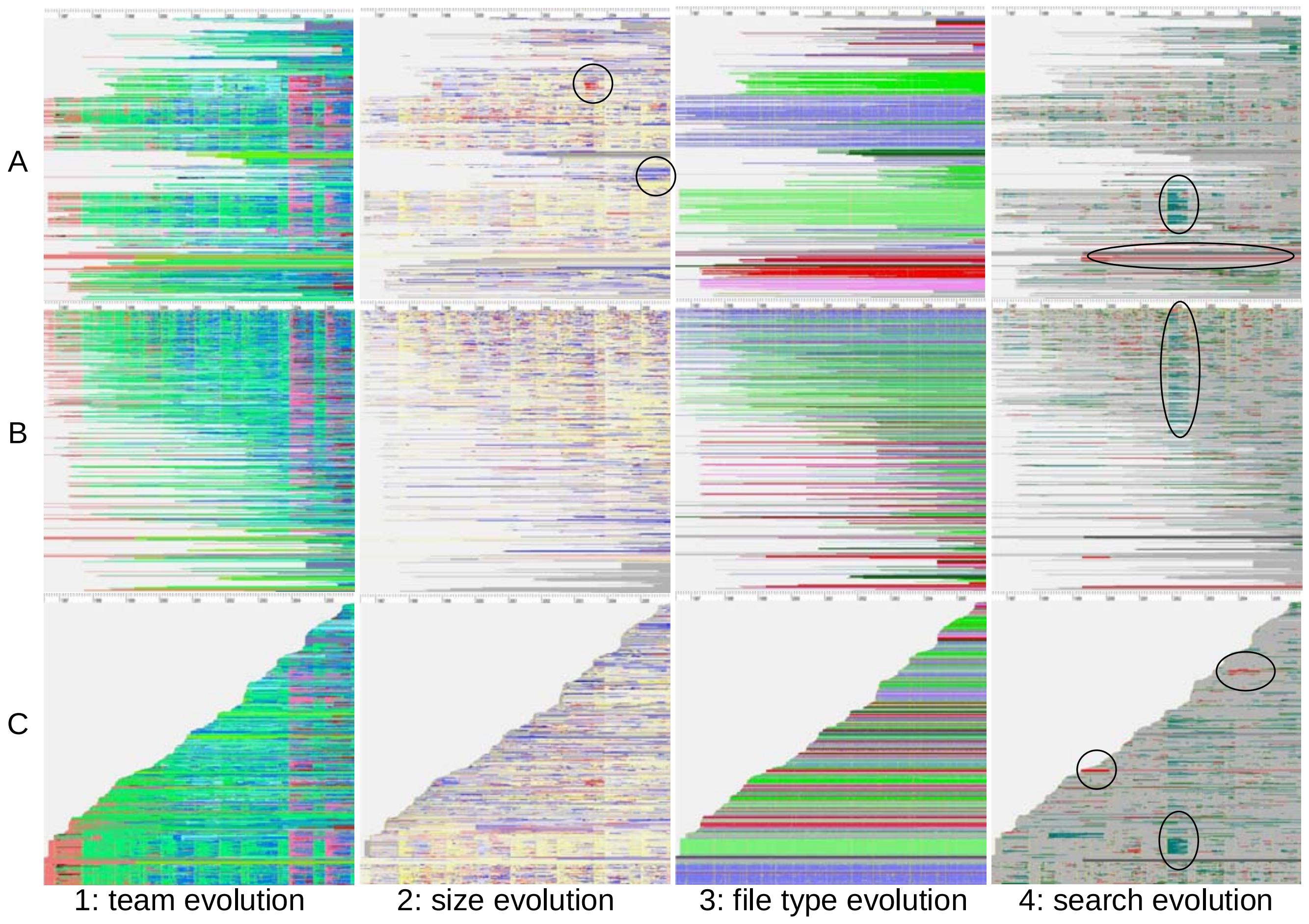

Figure 4: PostgreSQL metrics evolution visualization with CVSgrab

Figure 4 gives another example of CVS evolution visualization done using our proposed framework. It shows the evolution of PostgreSQL, an object-relational database management system project with a history of 10 years, 2829 files, and 27 authors. We used the same framework setup as in the previous example. The data acquisition step took 28 minutes: 7 minutes for the initial setup (i.e. one time retrieval of the last project version = 56MB) and 21 minutes for retrieving the evolution information to be visualized (29MB). The evolution retrieving time was in this case smaller than in the first example, even if more data was retrieved. This is explained by the connection overhead. When retrieving evolution data, the connection has to be established for each file. In this case the number of files was less than in the first example, which significantly improved the overall connection latency.

Figure 4 shows 12-snapshot matrix illustrating the evolution of PostgreSQL, structured similarly to Figure 3. Columns show the development team (1), size evolution (2), file type (3) and string search (4) encoded by colors, just as in example 1, except for file type and string search. In column 3, C source files are blue, light C headers are light green, SGML documentation files are normal green, SQL files are pink, and test support files are red. In column 4, green shows versions that contain the word "fix" in their associated commit comment, and red versions that contain the word "bug". As in the first example, the matrix rows use different sortings for arranging files on the vertical axis: alphabetical order (A), number of revisions (B) and creation time (C).

Assessing the evolution information depicted in Figure 4 one can compare the evolution of PostgreSQL with the one of ArgoUML presented in the first example, as follows. Cell C3 shows that the project started with a source code base (i.e. blue at the bottom) and not with a specification, as for ArgoUML. Even the header files containing interfaces were not fixed until a couple of months later (light green). As for ArgoUML, the initial contribution to the repository (C source and headers) was performed by one person (*scrappy* = red in C1) and incorporated many files (approx. 400). This suggests that previous developments existed that were not recorded in CVS. The rest of the evolution appears to be mainly the contribution of a few authors: *momjian* (light green), *tgl* (dark blue), *pgsql* (magenta), *petere* (cyan), *thomas* (yellow-greenish). The contributions of *momjian* and *tgl* are interleaved at periods of around 6-8 months (column 1) and address the most active parts of the system (B1). These parts correspond to the implementation files (i.e. C source code and headers), by correlation via A1 and A3. These parts are also targeted by *pgsql* in the last two project years. A detailed look at B1 and B2 reveals the contribution patterns of *momjian* and *tgl*. The versions committed by *momjian* do not usually bring changes in files sizes (i.e. they are yellow in B2) and are relatively done at large intervals. In contrast to this, the contributions of *tgl* are done at smaller intervals and cause often changes in the file size. Moreover, the contributions of *momjian* "interrupt" abruptly the ones of *tgl* but not conversely. This suggests the real work might be done by *tgl* while *momjian* has more the role of a code standard manager. A more in-depth investigation of the evolution using the details-on-demand mechanism of CVSgrab showed that the modifications done by *momjian* addressed mainly changes in indentation and copyright texts. A similar pattern holds for *pqsql*. Finally, p*etere* and *thomas* appeared to have mainly contributed to the system documentation (by correlating A1 and A3). As for ArgoUML, PostgreSQL seems to have a clean organization (A3): Source, header, documentation, and test files are well separated. Most of the activity takes place in the implementation files. Not only C files are modified but also headers and documentation files, which could suggest frequent architectural changes. No significant size modifications are registered throughout the project. The only exceptions, highlighted in A2, address the documentation part of the project. Finally, column 4 in Figure 4 shows the distribution of the words "fix" and "bug" along the project evolution. The green patterns highlighted in the image correspond to versions containing the word "fix". By correlating C3, C4 and C1, it seems that these patterns match header files in versions committed by *momjian*. Hence, it is possible they do not address important changes for the system functionality. Indeed, a more detailed analysis revealed that the word "fix" refers actually to a version of an indentation program used to format the text and not to the system code itself! Other occurrences of the word "fix" are evenly distributed largely over the evolution of C source files (A4). The red patterns highlighted in A4 and C4 correspond to versions containing the word "bug". They correspond to test files and appear towards the file creation moment. This, together with the fact that test files are created relatively early in the project, suggests an active test policy. The rest of the occurrences of the word "bug" are evenly distributed, mostly along the evolution of C source files.

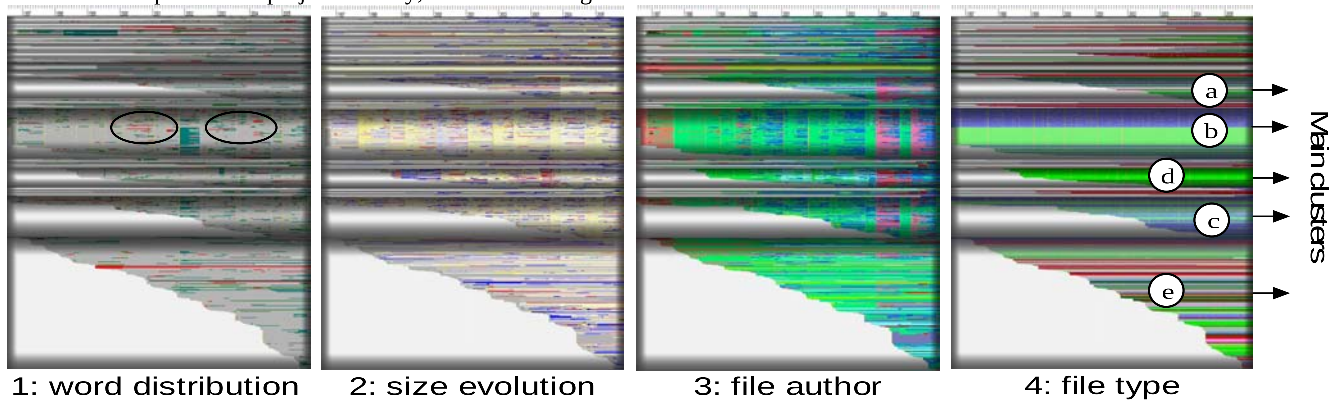

Figure 5: PostgreSQL evolution clusters visualization with CVSgrab

Error! Reference source not found. visualizes PostgreSQL evolution enriched with data analysis about clusters of files with common evolution. Four CVSgrab snapshots are presented. Clusters are encoded using plateau cushions. In each cluster, files are sorted in decreasing order of their creation time, from top to bottom. Color shows different file metrics: word distribution (1), size evolution (2), file author (3) and file type (4), as in Figure 4. There are mainly five important evolution groups. In column 4, one can see three main groups: source files (a,b,c), documentation (d), and test scenario files (e). We can easily see that source files introduced in the beginning of the project have a similar evolution (b). Hence, they may refer to a part of the system that has a high logical coupling and can be seen as a building block. The same holds for the other two clusters containing source code (a, c). All building blocks share the same developer network (3) and size evolution patterns (2). The block corresponding to the early introduced source code (b) has, however, a higher density of versions with comments containing the word "bug" (highlighted in 1). Hence, this block may contain a problematic implementation. Documentation forms a separate cluster (d), leading to the conclusion that it mainly targets the functionality of the system and not its detailed design, as it doesn't change in sync with the headers. Finally, the large cluster at the bottom of the images (e) corresponds to a miscellaneous collection of files including test scenarios. This cluster may thus refer to files intended to support the development process, rather than implementing real functionality.

Changing the cluster granularity level, one can further split the clusters presented above for a finer analysis of the system. This can be useful not only for performing a logical decomposition, but also for predicting future changes with different levels of confidence.

6. CONCLUSIONS

In this paper we propose a new framework for visual data mining of CVS software repositories. Our goal is twofold. On the one hand we aim to provide the research community with a base for experimentation of new techniques in data acquisition, analysis and visualization. On the other hand, we want to increase the framework acceptance by making it immediately available to the end users for CVS repository mining.

To achieve the first goal we propose a mediator module for CVS data acquisition that can easily integrate with current data extraction systems. The role of this module is to facilitate the resolution of CVS format incompatibility problems without requiring the modification / replacement of the data acquisition module. Secondly, we propose a new approach for quick visualization of data analysis results using the generic metric visualization mechanism of CVSgrab [17].

To make the framework immediately available to end users, we integrate the CVS mediator with a reference implementation of a data extraction tool. Additionally, we propose a new technique for identifying clusters of files with similar evolution. This could help users both to perform a logical decomposition of the system, and to predict future changes in the system from the perspective of select files. We integrate this technique as a data analysis module in the proposed framework, and we use CVSgrab [17] as visualization backend. Finally we illustrate the functionality of the integrated framework by visually mining the evolution of two industry-size Open Source projects: ArgoUML and PostgreSQL. The two cases demonstrate the framework has affordable time, bandwidth, and storage requirements for data acquisition. Additionally, it enables users to easily make complex evolution assessments by correlating evolution of multiple file metrics.

As a future direction of research we would like to improve the similarity measure of the evolution clustering mechanism by using additional attributes, e.g. file type, author. The challenge in this direction is to find the best similarity description that matches a given user requirement. Additionally, we would like to extend the framework with other generic visualization mechanisms, for easy assessment of data analysis techniques.

7. REFERENCES

[1] Ball, T., Kim, J.-M., Porter, A.A., and Siy, H.P. If your version control system could talk. *ICSE'97 Workshop on Process Modelling and Empirical Studies of Software Engineering*, May 1997. http://research.microsoft.com/~tball/papers/icse97-decay.pdf

[2] Bieman, J. M., Andrews, A. A., and Yang, H. J. Understanding change-proneness in OO software through visualization. *Proc. Intl. Workshop on Program Comprehension*, IEEE Press, 2003, pp. 44–53

[3] Bonsai online: http://www.mozilla.org/projects/bonsai/

[4] Collberg, C., Kobourov, S., Nagra, J., Pitts, J., and Wampler, K. A System for Graph-Based Visualization of the Evolution of Software. *Proc. ACM SoftVis'03*, ACM Press, 2003, pp. 77–86

[5] CVS online: http://www.nongnu.org/cvs/

[6] Eick, S.G., Steffen, J.L., and Sumner, E.E. Seesoft - A Tool For Visualizing Line Oriented Software Statistics. *IEEE Trans. on Software Engineering*, 18:11, IEEE Press, 1992, pp. 957– 968

[7] Fischer, M., Pinzger, M., and Gall, H. Populating a Release History Database from version control and bug tracking systems. *Proc. Intl. Conf. on Software Maintenance*, IEEE Press, 2003, pp. 23–32

[8] Froehlich, J., and Dourish, P., Unifying Artifacts and Activities in a Visual Tool for Distributed Software Development Teams. *Proc. ICSE'04*, IEEE Press, 2004, pp.387–396

[9] Gall, H., Jazayeri, M., and Krajewski, J. CVS release history data for detecting logical couplings. *Proc. IWPSE'03*, IEEE Press, 2003, pp. 13–23

[10] German, D., and Mockus, A. Automating the measurement of open source projects. *ICSE '03 Workshop on Open Source Software Engineering, Automating the Measurement of Open Source Projects*, http://www.research.avayalabs.com/user/audris/papers/oose03.pdf

[11] German, D., Hindle, A., and Jordan, N. Visualizing the evolution of software using softchange. In *Proc. Intl .Conference on Software Engineering and Knowledge Engineering* (SEKE'04), pp. 336–341

[12] Lanza, M. The evolution matix: Recovering software evolution using software visualization techniques. In *Proc. Intl. Workshop on Principles of Software Evolution*, ACM Press, 2001, pp. 37–42

[13] Lopez-Fernandez, L., Robles, G., and Gonzalez-Barahona, J.M. Applying Social Network Analysis to the Information in CVS Repositories. *Intl. Workshop on Mining Software Repositories (MSR)*, 2004, http://opensource.mit.edu/papers/llopez-sna-short.pdf

[14] NetBeans.javacvs online: http://javacvs.netbeans.org/

[15] Subversion online: http://subversion.tigris.org/

[16] Voinea, L., Telea, A., and van Wijk, J.J. CVSscan: Visualization of code evolution. *Proc. ACM SoftVis*, ACM Press, 2005, pp. 47 – 56

[17] Voinea, L., and Telea, A. CVSgrab: Mining the History of Large Software Projects. *Proc. EuroVis'06*, IEEE Press, 2006.

[18] Wu, K., Spitzer, C.W., Hassan, A.E., and Holt, R.C. Evolution Spectrographs: Visualizing Punctuated Change in Software Evolution. In *Proc. Intl. Workshop on Principles of Software Evolution* (IWPSE'04), IEEE Press, 2004, pp. 57-66

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

[20] Ying, A.T.T., Murphy, G.C., Ng, R., Chu-Carroll, M.C., Predicting Source Code Changes by Mining Revision History. *IEEE Trans. on Software Engineering*, 30:9, IEEE Press, 2004, pp. 574-586

[21] Zimmermann, T., Weißgerber, P., Diehl, S., Zeller, A., Mining version histories to guide software changes. *Proc. Intl. Conference on Software Engineering* (ICSE), IEEE Press, 2004, pp. 429–445

[22] Zimmermann, T., Weißgerber, P., Preprocessing CVS Data for Fine-grained Analysis. *Intl. Workshop on Mining Software Repositories (MSR)*, May 2004, http://www.st.cs.uni-sb.de/papers/msr2004/msr2004.pdf